## إقـــــرار

**أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:**

أقر بأن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، وإن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل لنيل درجة أو لقب علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

## DECLARATION

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification

Student's name: Ramzi A. Matar اسم الطالب: رمزي عاطف مطر

Signature: التوقيع:

Date: التاريخ: 2015/09/21

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Islamic University – Gaza
Deanship of Graduate Studies
Faculty of Information Technology

الجامعة الإسلامية – غزة
عمـــادة الدراسـات العليـا
كلية تكنولوجيا المعلومات

# *Network Intrusion Detection Using One-Class Classification Based on Standard Deviation of Service's Normal Behavior*

A Thesis Submitted in Partial Fulfillment of the Requirement
for the Degree of Master in Information Technology

**Prepared by:**
Ramzi A.M. Matar
120130295

**Supervised By:**

Dr. Tawfiq S. Barhoom

*2015AD/ 1436AH*

بسم الله الرحمن الرحيم


الجامعة الإسلامية – غزة
The Islamic University - Gaza

مكتب نائب الرئيس للبحث العلمي والدراسات العليا هاتف داخلي: 1150


الرقم: ج س غ/35/ ..... Ref

التاريخ 2015/09/08 ..... Date

# نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحث/ **رمزي عاطف محمد مطر** لنيل درجة الماجستير في **كلية *تكنولوجيا المعلومات*** **برنامج تكنولوجيا المعلومات** وموضوعها:

## كشف التسلل للشبكة باستخدام التصنيف أحادي الفئة المعتمد على الإنحراف المعياري للسلوك الطبيعي للخدمة

**Network Intrusion Detection Using One-Class Classification Based on Standard Deviation of Service's Normal Behavior**

وبعد المناقشة التي تمت اليوم الثلاثاء 24 ذو القعدة 1436هـ، الموافق 2015/09/08م الساعة الحادية عشرة صباحاً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

| | | |
|---|---|---|
| **د. توفيق سليمان برهوم** | **مشرفاً و رئيساً** | ..... |
| **أ.د. علاء مصطفى الهليس** | **مناقشاً داخلياً** | ..... |
| **د. ناجي شكري الظاظا** | **مناقشاً خارجياً** | ..... |

وبعد المداولة أوصت اللجنة بمنح الباحث درجة الماجستير في **كلية *تكنولوجيا المعلومات*/ برنامج تكنولوجيا المعلومات.**

***واللجنة إذ تمنحه هذه الدرجة فإنها توصيه بتقوى الله ولزوم طاعته وأن يسخر علمه في خدمة دينه ووطنه.***

والله ولي التوفيق،،،

**نائب الرئيس لشئون البحث العلمي والدراسات العليا**

**أ.د. عبدالرؤوف علي المناعمة**


P.O. Box 108, Rimal, Gaza, Palestine fax: +970 (8) 286 0800 فاكس Tel: +970 (8) 286 0700 هاتف ص.ب. 108 الرمال، غزة، فلسطين
public@iugaza.edu.ps www.iugaza.edu.ps

## *Dedication*

*To My Father, and Mother,*

*To My Wife,*

*To My Children,*

*To My Sisters and Brothers,*

*To My Friends,*

*To those who gave me all kinds of support,*

*To all, I dedicate this work.*

## *Acknowledgements*

*Praise is to Allah,*

*First and foremost, I wish to thank Allah for giving me strength and courage to complete this thesis and research. I would like to express my gratitude to my supervisor Dr. Tawfiq Barhoom, for giving me the opportunity to work with him and guiding and helping me throughout this research and other courses.*

*Thanks to my parents who have given credit after God in all things.*

*Thanks a lot to my wife who has supported me throughout my study.*

*Special thanks to Alaqsa University which gave me the allowance to collect the real traffic of its web server and also providing me servers for performing the attack experiments. Special thanks to Eng. Hassan Dawoud, the manager of Networks and Communications department, and to Eng. Mohammed Almusaddar the head of Networks and Internet department.*

*Thanks to my friends who encouraged me to make every effort and determination, Last but not least, I would like to thank my family members.*

# كشف التسلل للشبكة باستخدام التصنيف أحادي الفئة المعتمد على الانحراف المعياري للسلوك الطبيعي للخدمة


## الملخص

لازال استخدام شبكات الحاسوب والإنترنت على نحو متزايد في حياتنا اليومية. فبسبب النمو الهائل للهجمات على الشبكة، أصبحت أنظمة كشف التسلل للشبكة (NIDS) عنصرا أساسيا حيث تلعب دوراً حيويا لأمن شبكات الحاسوب لحماية موارد الشبكة من أي دخول غير مصرح به قد يجمع بيانات سرية، أو يؤثر على توفر الخدمة أو تنتهك سلامة البيانات. الكثير من الجهود البحثية بذلت نحو تصميم NIDS مثالية لديها معدل اكتشاف عالي ومعدل إنذار كاذب منخفض. البعض استخدم أساليب تقليدية "الكشف عن سوء استخدام" المعتمد على المعرفة المسبقة للهجوم، إلا أن هذه الأساليب تفشل في الكشف عن الهجمات الغير معروفة ولهذا هناك حاجة ملحة لتقنيات بديلة لكشف التسلل.

العديد من الباحثين لا يزالون يحاولون حل المشكلة باستخدام تقنيات التعلم بالآلة مثل التعلم المسمى والغير المسمى ؛ مشكلة استخدام التعلم المسمى تكمن في تكلفة إنتاج بيانات مسماة والتي هي أمر ضروري لتدريب النموذج، بالإضافة إلى أن النموذج يتدرب على هجمات معروفة والتي من الممكن أن تفشل في معرفة هجمات مختلفة جديدة. من ناحية أخرى، يواجه التعليم الغير مسمى مشكلة تسمية المجموعات المولّدة؛ أيا من المجموعات طبيعية أو غير طبيعية. تعاني أساليب التعليم شبه المسمى من قيود تمنعها من التفوق على التعليم المسمى ما لم يكون المحلل متأكد تماماً من وجود بعض العلاقات غير بديهية بين التوزيع المسمى والغير المسمى. بسبب القيود الموجودة في التقنيات السابقة ونظراً لتنوع وتعدد أنماط هجمات الشبكة، تم استخدام تقنية تعلم رابعة تدعى التصنيف أحادي الفئة (OCC) لتعلم سلوك فئة واحدة، وهي بالعادة السلوك الطبيعي، لاكتشاف أي انحراف عن هذه الفئة. ولكن عند تطبيق هذه التقنية على الشبكة ككل فإنها تعاني من تعدد عالي لميزات شبكية متعددة الأبعاد. وأيضاً قد تنشأ مشاكل عند وجود اختلافات كبيرة في الكثافة. للتغلب على هذه المشاكل تم اقتراح نموذج OCC-NIDS المعتمد على الانحراف المعياري للسلوك الطبيعي للخدمة.تعاملنا من خلال هذا النموذج مع كل خدمة في الشبكة كفئة واحدة بدلا من التعامل مع جميع خدمات الشبكة كفئة واحدة.من خلال هذا النهج استخدمنا الميزات ذات الصلة بكل خدمة على حدة، بالتالي تم تقليل التعدد العالي لميزات الشبكة وأيضاً التأكد من أن كل فئة لها على نحو تقديري توزيع موحد في الكثافة.

قمنا بتقييم النموذج الأولي المقترح على بيانات حقيقية وعلى بيانات تدعى KDD Cup'99 مشهورة. أثبت النموذج المقترح على أنه قادر على كشف حركة الشبكة الغير طبيعية بمعدل اكتشاف عالي وإنذار كاذب منخفض، حيث حقق النموذج المقترح نسبة 98.14% معدل اكتشاف ومعدل دقة 98.79% ومعدل إنذار كاذب 0.13% على البيانات الحقيقية. بينما باستخدام بيانات KDD Cup'99 حقق 99.88% معدل اكتشاف ومعدل دقة 99.6% مع إنذار كاذب وصلت إلى 0.77 %.

# Abstract

Computer networks and internet have been increasingly used in our daily life. Due to the explosive growth of network attacks, network intrusion detection systems (NIDS) have become an essential network component which plays a vital role for computer networks' security. The main purpose of NIDS is to protect network resources from any unauthorized access that may gather confidential data, affect its availability or violate its data integrity. A lot of efforts have been given toward designing a perfect NIDS that has a high detection rate and low false alarm rate. Some have used misuse detection technique which fails to detect zero-day attacks, such that there is a high demand for alternative detection techniques.

The problems of using supervised learning is the cost of producing labeled dataset, and also the model is trained on known attacks which may fail to detect new variant attacks. On the other hand, unsupervised learning has the problem of labeling the generated clusters; which cluster is normal or abnormal. Semi-supervised learning techniques suffers from the limitation that it cannot outperform supervised classification unless the analyst is absolutely certain that there is some nontrivial relationship between labeled and the unlabeled distribution. Because of the limitations of previous learning techniques, and because of the increasing diversity and polymorphism of network attacks, a fourth learning technique called One-Class Classification (OCC) has been used to learn the behavior of single class, which is commonly normal traffic, to detect any deviation from it. However when applying this technique on network as a whole it suffers from the high dimensional network feature spaces. Also, problems may arise when large differences in density exist. To overcome these problems, we proposed a primary OCC-NIDS model based on the standard deviation of service's normal behavior. Through this model we dealt with each network service as single class instead of dealing with all network services as a single class. By this way we use just the relevant features of each service, hence reducing the high dimensional network feature spaces and also ensure that each class has - a proximately - uniform distribution.

We evaluated the proposed primary model on our testbed dataset and on KDD Cup'99 datasets. The proposed model proved that it has the ability to detect abnormal network traffic with high detection rate and low false positive rate. Our proposed model achieved 98.14% detection rate and 98.74% accuracy rate with 0.13% false positive rate on our testbed dataset. While on KDD Cup'99 dataset our model achieved 99.88% detection rate and 99.6% accuracy rate with a false alarm rate reached 0.77% and false positive rate 0.028%.



**Keywords**

Network Intrusion Detection, Anomaly detection, One-Class Classification learning, Standard Deviation, Service's Normal Behavior.

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviation

| | |
|---|---|
| BM-AUN2015 | Barhoom and Matar - Alaqsa University Network collected in 2015 |
| CIA | Confidentiality, Integrity, Availability |
| DDoS | Distributed Denial of Service |
| DMZ | Demilitarized Zone |
| DoS | Denial of Service |
| ECR_I | Echo/Replay service in ICMP. |
| FCM | Fuzzy C-Means |
| FN | False Negative |
| FP | False Positive |
| HIDS | Host-based Intrusion Detection Systems |
| HTTP | Hyper Text Transfer Protocol |
| ICMP | Internet Control Message Protocol |
| IDS | Intrusion Detection Systems |
| IP | Internet Protocol |
| KDD | Knowledge Discovery and Data Mining |
| KNN | K-Nearest-Neighbor |
| LOF | Local Outlier Factor |
| LoOP | Local Outlier Probability |
| NIDS | Network-based Intrusion Detection Systems |
| OCC | One-Class Classification |
| OCC-NID | One-Class Classification Intrusion Detection Systems |
| OCSVM | One Class Support Vector Machine |
| POP3 | Post Office Protocol v3 |
| R2L | Unauthorized access from a remote machine |
| SD | Standard Deviation |
| SMOTE | Synthetic Minority Over-sampling Technique |
| TCP | Transmission Control Protocol |
| TN | True Negative |
| TP | True Positive |
| U2R | Unauthorized access to local super user (root) privileges |
| UDP | User Datagram Protocol |

# Chapter 1: Introduction

In the modern life, information technology and communications infrastructure play a critical role in people's life. The Internet connects thousands of sub-networks and thereby links over 1billion computers worldwide [1]. The variety of attacks affected computers linked to the Internet, ranging from zero-day exploits crafted for stealthy compromises to computer worms capable of mass-infections. These attacks put both personal as well as business computer systems at risk to be remotely compromised and misused for illegal purpose, (e.g., gathering of confidential data, affecting services availability or violating data integrity) which are the three main components of computer security known as confidential, Integrity, Availability (CIA) Triad [2].

There are two main problems that cause the increase of networks attacks: First, there is a deficit of security awareness in software development [3], (e.g. existing of bugs which make it a vulnerable for attacks exploitations like stake-overflow). A second reason is due to the increasing automation and sophistication of network attacks [4]. A widespread availability of generic attack tools that have an amazing range of functionality, including network surveillance, polymorphic shellcodes and distributed propagation. As an example, the computer worm "Slammer" possess the ability to infect thousands of hosts in a couple of minutes [5]. Such capabilities make malicious software and network attacks attractive for illegal business, as they allow for abuse of millions of computer systems. Due to the explosive growth of the network attacks, intrusion detection systems have become an essential network component which plays a vital role for computer networks and security.

## *1.1 Intrusion Detection*

Intrusion detection is a branch of computer security originating from a research on securing multi-user systems [6]. Formally, computer security deals with the protection of the confidentiality, integrity and availability of resources [7]. Thus, the definition of intrusion/attack in terms of these aspects is any action or set of actions that are attempt to compromise the integrity, confidentiality or availability of a resource [8].

There are three security layers to defeat computer attacks [9], the first layer is the prevention of attack which aims to block any unauthorized access to the network and systems, e.g. by means of cryptography or access privileges. However, based on the history of security violations (e.g. gaining access using stack-overflow like worms attacks), it is impossible to block all types of malicious activities Thus, there's a need for a detection of attacks which is the goal of the second layer, this layer is known as intrusion detection system that detect the attack and send an alarm to the administrator or respond to the attack according to the predefined rules. The last layer is known as a recovery layer which aims to alleviate potential damage of attacks and ease removal of existing vulnerabilities. Our goal is to design Intrusion Detection System (IDS) which is capable to identify unknown attacks that slipped through a

preceding prevention layer and thus integrates into the depicted cycle at the detection layer.

IDSs are considered to act as the second defense line against network attacks that failed to be addressed by preventive mechanisms [10]. An Intrusion detection system is defined in [11] as "*A system that dynamically monitors the events taking place on a system and decides whether these events are symptoms of an attack or constitute a legitimate use of the system*".

The intrusion detection system gains its dynamism through its ability of generating real-time network instances as shown in Figure 1.1. There are four main stages in intrusion detection system, (1) packets sniffing (capturing), (2) network instance generation from raw packets, (3) classifying the generated instance into either normal or abnormal instance and (4) notifying the network administrator about any abnormal behavior..

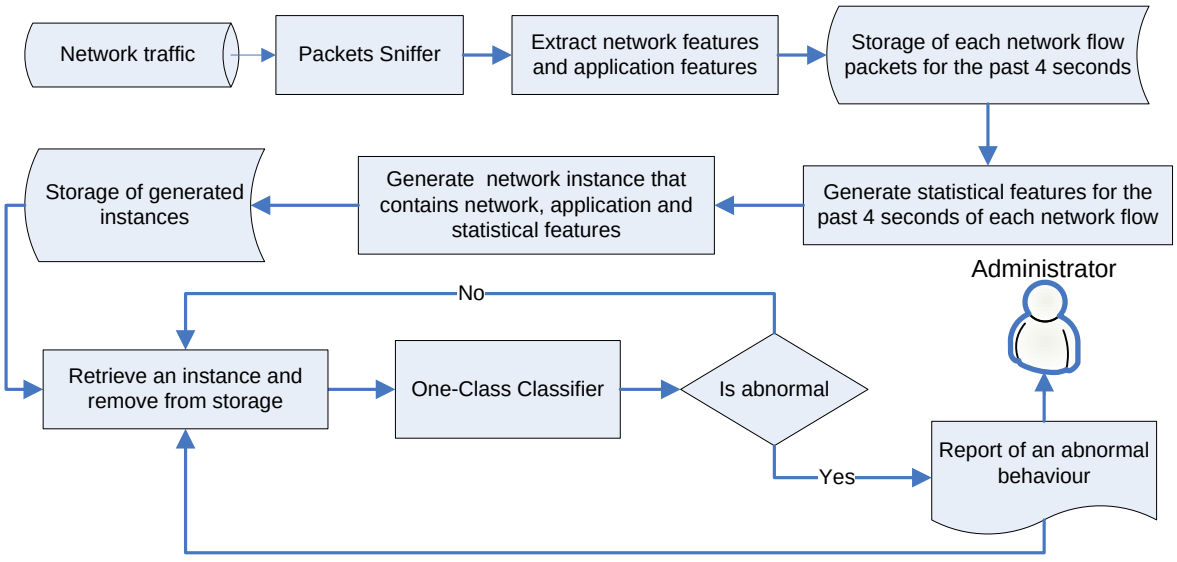


**Figure 1.1 An Overview of all components that make NIDS dynamic**

As shown in Figure 1.1 the packet sniffer captures packets from the network. These packets are the processed to extract its network, application and statistical features. The generated network instances are stored for classification purposes. In the classification stage, the classifier classifies all the stored instances into either normal or abnormal, and if it is abnormal, the network administrator is reported about this instance.

## *1.2 Standard Deviation*

The Standard Deviation (SD) (represented by the Greek letter sigma, σ which is the square root of the variance $\sigma^2$) measures the amount of variation or dispersion from the average [12]. A low standard deviation indicates that the data points tend to be very close to the mean (also called expected value); a high standard deviation indicates that the data points are spread out over a large range of values. The variance $\sigma^2$ is the average of the squared differences from the Mean. There are two formulas to calculate the standard deviation. The "Population Standard Deviation", which is used when we have a complete dataset and the

"Sample Standard Deviation", used when we have a sample dataset. In our proposed method for calculating the standard deviation we used the sample standard deviation because we don't have the a complete normal data. The sample standard deviation is shown in Eq 1.1.

$$s = \sqrt[2]{\frac{1}{N-1}\sum_{i=1}^{N}(xi-\mu x)^2} \quad \text{---------- (Eq. 1.1) [12]}$$

## *1.3 One-Class Classification*

One-Class Classification (OCC) term was coined by Moya and Hush (1996) [13]. It was proposed to solve the problem of conventional multi-class classification algorithms that aim to classify an unknown object into one of several pre-defined categories. A problem arises when the unknown object does not belong to any of those categories[14]. In OCC [15], one of the classes (referred to as the positive class or target class) is well characterized by instances in the training data. For the other class (non-target), it has either no instances at all, very few of them, or they do not form a statistically-representative sample of the negative concept. A motivation example to the importance of one-class classification, let us consider the following scenario: OCC can be relevant in detecting machine faults, for instance. A classifier should detect when the machine is showing abnormal/faulty behavior. Measurements on the normal operation of the machine (positive class training data) are easy to obtain. On the other hand, most faults will not have occurred so one will have little or no training data for the negative class.

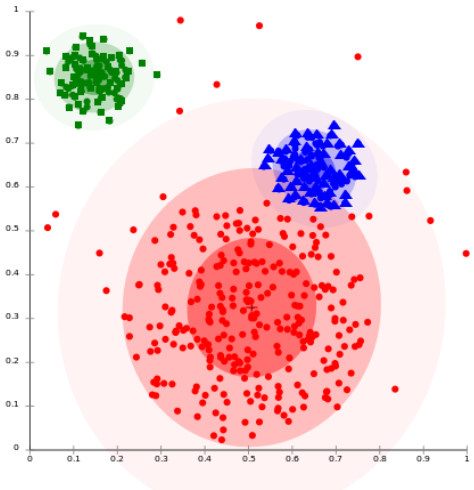

**Figure 1.2 An illustration of intrusion detection using standard deviation [16]**

The bold-circle instances is the normal behavior's class with standard deviation from center to the first inner circle.

The task in OCC is to define a classification boundary around the positive (or target) class, such that it accepts as many objects as possible from the positive class, while it minimizes the chance of accepting non-positive (or outlier) objects [17]. As illustrated in Figure 1.2, there

are 3 classes, the class with bold-circle instances, which is the biggest scattered one, is the normal class and the others, with bold-square and bold-triangle instances, are the abnormal instances. It is also harder to decide which attributes should be used to find the best separation of the positive and non-positive class objects. In particular, when the boundary of the data is long and non-convex, the required number of training objects might be very high. [14]. The limitations of this technique is that it often require a large data set to determine the boundary accurately [14]. Another limitation exists in high dimensional feature spaces, at which the OCC becomes very inefficient [14] where in OCC we have one class, so the deviation from this class must be measured using one feature space. Also, problems may arise when large differences in density exist. Objects in low-density areas will be rejected although they are legitimate objects [14].

## *1.4 Research Motivation*

Due to the explosive growth of network attacks, network intrusion detection systems (NIDS) have become an essential network component which plays a vital role for computer networks' security to protect network resources from any unauthorized access that may gather confidential data, affect its availability or violate its data integrity. A lot of efforts have been given toward designing a perfect NIDS that has a high detection rate and low false alarm rate. Some [18, 19] have used misuse detection technique which fails to detect zero-day attacks, such that there is a high demand for alternative detection techniques that can reach a high detection rate and low false alarm.

## *1.5 Statement of the Problem*

Current NIDSs, which based on One Class Classification (OCC) learning technique, suffer from the high dimensional network feature spaces. It also suffers from the existence of large differences in density which affect the detection accuracy. These drawbacks arise because of applying OCC on network instances as a whole and deal with it as a single class.

## *1.6 Research Objectives*

The objectives of our research are to overcome the limitations of applying datamining, classification or clustering, techniques. These limitations arise because of the increasing diversity and polymorphism of network attacks that have become a great challenge which obstruct modeling signatures to be used with misuse detection, or labeling of instances to be used with supervised learning or labeling the generated clusters. And because the current semi-supervised learning techniques suffers from the limitation that it cannot outperform supervised classification when the assumptions made do not hold which make it worse than supervised learning.

### 1.6.1 Main Objective

The main objective of this research is to overcome the drawbacks of current NIDSs which use OCC learning technique. These drawbacks are the high dimensional network feature spaces, and the existence of large differences in density which affect the detection accuracy. We proposed a primary OCC model based on the standard deviation of service's normal behavior.

### 1.6.2 Specific Objectives

The specific objectives of the project are:

- Collect labeled real network traffic datasets that have both normal and attack instances.
- Separate normal instances in different classes based on service used to build an OCC model for each class.
- Perform feature selection for each service to get the most relevant service's features space in order to reduce high dimensional network feature spaces.
- Build an OCC model based on the standard deviation of service's normal behavior for each service to be able to classify the traffic data into normal or abnormal.
- Test the OCC model on our real dataset called BM-AUN2015 in addition to a benchmark dataset to observe the system ability to detect new attacks and store the experimental results for evaluation.
- A Prof of concept Evaluation of the OCC model using confusion matrix to get acknowledgment about its detection accuracy.
- Compare the results obtained from OCC models which performed on KDD Cup'99 dataset with previous related works models used this dataset in order to be sure that our model has achieved its main objective.

## *1.7 Significance of the Research*

- Build NID model that can detect unseen before attacks.
- Allow the network administrator to choose the acceptable false alarm rate for each service in order to optimize the overall detection rate.
- This research can be extended to increase the detection rate and reduce the false alarm rate by including the service payload's feature space, where each service has its own payload feature space.
- The detection time is reduced using our model; instead of measuring the distance between a new instance and the normal class which include all the network feature spaces, we measure the distance between a new service instance with its corresponding service class which has its own feature space.

## *1.8 Scope and Limitations of the Research*

In this research we built OCC model which is based on the availability of enough normal instances dataset. The detection process is done by applying a new OCC learning technique based on the standard deviation of each services' normal behavior. Our primary model that we built has some limitations and assumptions such as:

- The primary model is evaluated on a real dataset called real dataset BM-AUN2015, and also on KDD Cup'99 [20] which is a benchmark dataset.
- The primary model depends mainly on the existence of enough normal instances for each service to be built in order to have an accurate standard deviation.
- The primary model built based on the assumption that most of the computers involved in the network data collection process are not infected.
- The primary model doesn't classify attacks based on attack categories instead of that it will notify the administrator about the source of any abnormal behavior in the network and provide him information about that abnormal instance.

## *1.9 Outline of the Thesis*

This thesis is divided into seven chapters, which are structured around the objectives of the research. The thesis was organized as follows:

**Chapter 1**, in this chapter, intrusion detection system, abnormal definition, main goals for detection model, the research statement problem, objectives and outlines were identified.

**Chapter 2**, in this chapter, literature review such as identifying anomalies, types and characteristics were presented. Also intrusion detection techniques, supervised and supervised learning, data mining techniques, OCC techniques which used in the model were defined.

**Chapter 3**, in this chapter, the related works which used data mining classification/clustering techniques for detection of network intrusions were presented and discussed. Besides, the main advantages and shortages were highlight and discussed.

**Chapter 4**, in this chapter, the real environment of the real dataset was described, the tools and the attack testing tools were also explained. Statistical graphs and collection scenarios were presented in details with explanation.

**Chapter 5**, in this chapter, the proposed model and methodology was presented. The model architectures and scenarios were also presented. There is explanation about our data sets used, dataset preprocessing, construct behavior rules, instance identification method. There are baseline experiments to choose every parameter, tools used in the model.

**Chapters 6**, in this chapter, the details of experiments were presented, analyzed the results, discussed each experiment, and drew main figures and summaries.

**Chapter 7**, in this chapter, the conclusion and summary of the research achievement of experiments were presented. Finally, future work was suggested.

# Chapter 2: Theoretical Background

In this chapter, anomalies, types, characteristics, intrusion detection techniques, supervised and unsupervised learning definition and data mining classification and clustering techniques were explained.

## *2.1 Intrusion Definition, Types and Detection*

An intrusion is defined by Heady et al. [21] as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource.

Attacks can be classified into two major types; attacks based on goal and attacks based on protocol used. Figure 2.1 shows the hierarchy chart of network attacks. In the following subsections, we presented the attack types and gave a brief description of the attacks of each type which we used in training and testing of our model.

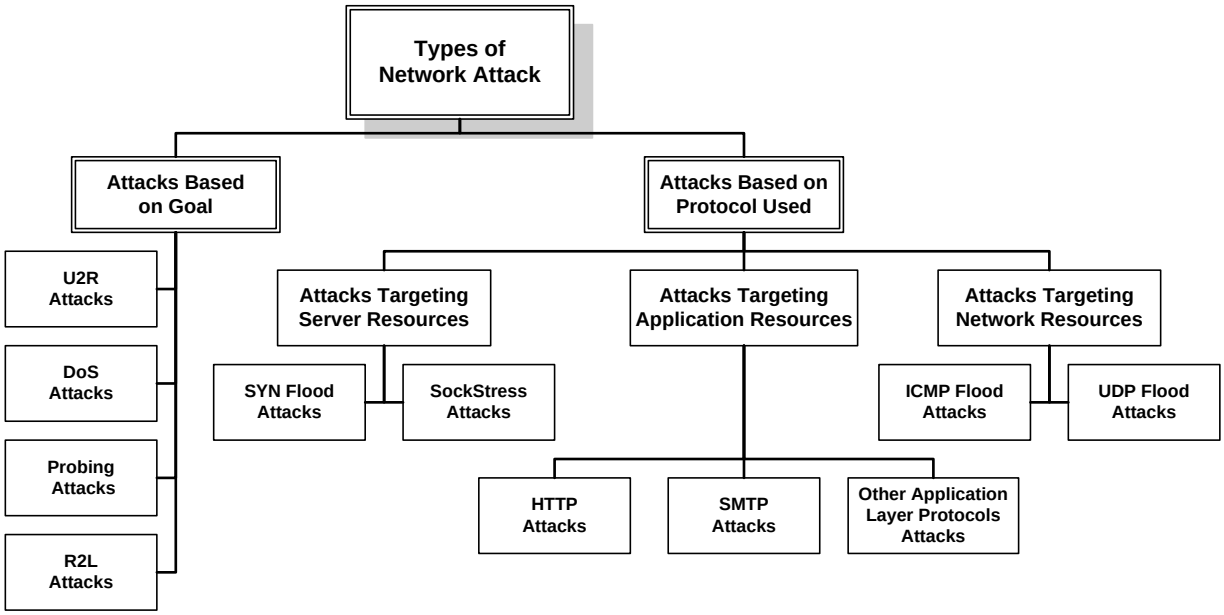


**Figure 2.1 Types of network attacks**

### 2.1.1 Types of Attack Based on Goal

Despite the fact that many researchers attempted to classify the computer attacks into suitable categories, there is still no adopted standard threats and attacks classification [22] .The most widely used classification for attacks in the research communities is the one adopted by [23]. This classification categorizes computer attacks, as shown in Figure 2.1, into:

1- DoS: Denial of service – where an attacker tries to prevent legitimate users from using a service. e.g. Syn flooding

2- Probing: Surveillance and other probing, where an attacker tries to gain information about the target host., e.g. port scanning.

3- U2R: unauthorized access to local super user (root) privileges, where an attacker has local access to the victim machine and tries to gain super user privileges., e.g. buffer overflow attacks.
4- R2L: unauthorized access from a remote machine, where an attacker does not have an account on the victim machine, hence tries to gain access., e.g. password guessing.

### 2.1.2 Types of Attack Based on Protocol Used

Attacks is also classified based on exploited protocol into three main categories (1) Attacks targeting network resources, (2) Attacks Targeting Server Resources and (3) Attacks Targeting Application Resources [24, 25], as shown in Figure 2.1.

#### 2.1.2.1 Attacks Targeting Network Resources

Attacks that target network resources attempt to consume all of a victim's network bandwidth by using a large volume of illegitimate traffic to saturate the company's Internet pipe. Attacks of this manner, called network floods, are simple yet effective. In a typical flooding attack, the offence is distributed among an army of thousands of volunteered or compromised computers – a botnet – that simply sends a huge amount of traffic to the targeted site, overwhelming its network. This type of attack is also known as network layer attack [26, 27].

**UDP Flood attack** [25] does not exploit a specific vulnerability, but rather simply abuses normal behavior at a high enough level to cause network congestion for a targeted network. It consists of sending a large number of UDP datagrams from potentially spoofed IP addresses to different ports on a victim server; the server receiving this traffic is unable to process every request, and consumes all of its bandwidth attempting to send ICMP "destination unreachable" packet replies to confirm that there was no application listening on the targeted ports. User Datagram Protocol (UDP) is a connectionless protocol that uses datagrams embedded in IP packets for communication without needing to create a session between two devices [RFC 768].

**ICMP Flood** [25], also known as Ping Flood, is a non-vulnerability based attack; that is, it does not rely on any specific vulnerability to achieve denial-of-service. An ICMP Flood can involve any type of ICMP message of echo request; once enough ICMP traffic is sent to a target server, it becomes overwhelmed from attempting to process every request, resulting in a denial-of-service condition. Internet Control Message Protocol (ICMP) is another connectionless protocol used for IP operations, diagnostics, and errors [RFC 777].

#### 2.1.2.2 Attacks Targeting Server Resources

Attacks that target server resources attempt to exhaust a server's processing capabilities or memory, potentially causing a denial-of service condition. The idea is that an attacker can take advantage of an existing vulnerability on the target server (or a weakness in a communication protocol, e.g. TCP protocol) in order to cause the target server to become busy handling illegitimate requests so that it no longer has the resources to handle legitimate ones. This type of attack is also known as network layer attack [26, 27].

**SYN Flood attack** is based on exploiting the standard TCP three-way handshake [28]. The TCP three-way handshake requires a three-packet exchange to be performed before a client can officially use the service. A server, upon receiving an initial SYN (synchronize/start) request from a client, sends back a SYN/ACK (synchronize/acknowledge) packet and waits for the client to send the final ACK (acknowledge) as shown in Figure 2.2.

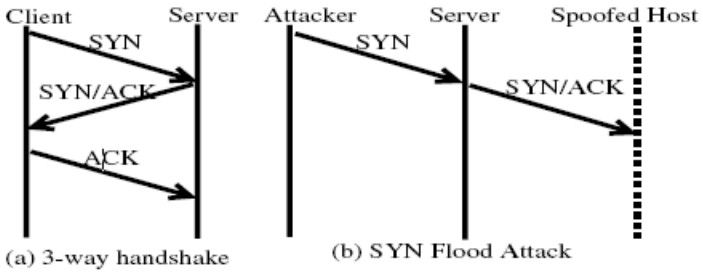


**Figure 2.2 Overview of a 3-way handshake and a SYN Flood attack [29]**

However, it is possible that the client send more of initial SYN's without sending the corresponding ACK's, essentially leaving the server waiting for the non-existent ACK's. Considering that the server only has a limited buffer queue for new connections, SYN Flood results in the server being unable to process other incoming connections as the queue gets overloaded.

**Sock Stress attack** differs than SYN flood attack that it complete the three-way handshake required for open TCP session, but the third ACK packet sent by the client has a TCP window size equals zero [30]. Now the server will have to "probe" the client until the zero window opens up as shown in Figure 2.3. Through this technique, the attacker bypass IDS through a legitimate TCP connection. The result is similar to a connection flood, except that the sockets remain open potentially indefinitely and the attacker can in an interval of time to send a HTTP payload for example, if he connect to a web server. The Widow Size field in TCP protocol indicates how much more room is in the buffer in each point of time. Window size set to zero means that there is no more space whatsoever and that the other side should stop sending more data until further notice. [RFC 2581].

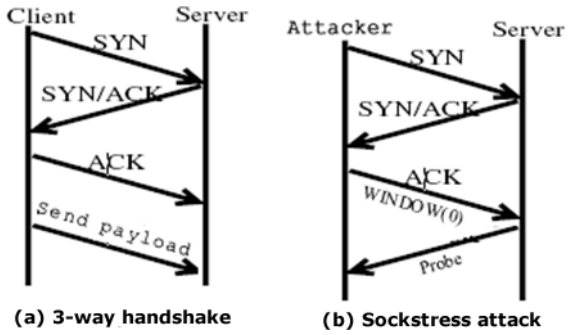


**Figure 2.3 Overview of a 3-way handshake and a Sock stress attack**

### 2.1.2.3 Attacks Targeting Application Resources

These attacks are known as application layer attacks [26, 27], the attacker exploits the

vulnerability of application layer protocols such as HTTP, POP3, SOAP, FTP, etc... The attack categories exploit this layer are DoS, DDoS R2L and U2R attacks. As an example of HTTP attacks, a DoS attacks like SlowRead, cross-site scripting (XSS) and SQL-Injection which are R2L. Note that the requester couldn't use spoofed IP addresses in this type of attacks [26]. In our real dataset collected, described in chapter 4, we considered to perform Slowloris, Slowpost, and Slowread attacks as an application layer attacks that attack a web server.

**Slowloris attack** [31] also known as slow headers attack which is an HTTP get-based attack that can bring down a Web server using a limited number of machines or even a single machine. The attacker sends partial HTTP requests, not a complete set of request headers, that continuously and rapidly grow, slowly update, and never close as shown in Figure 2.4. The server in this case waiting for a blank line (CRLF: Carriage Return Line Feed) which indicate that the HTTP Header end. The attack continues until all available sockets are taken up by these requests and the Web server becomes inaccessible. The attacker aims to keep the socket opened as long as possible, while he opens many connections to the server to do a DoS attack.

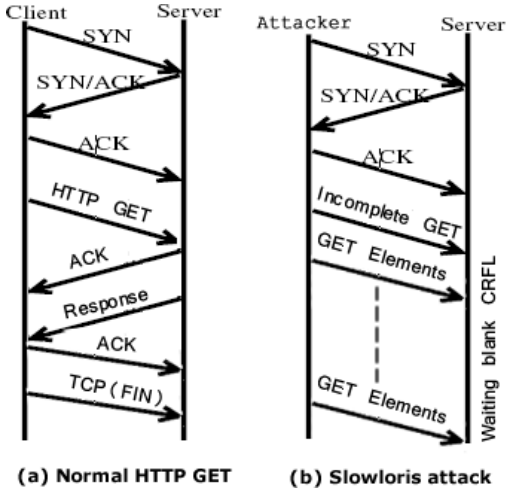


**Figure 2.4 Overview of a normal HTTP GET and a Slowloris attack**

**Slowpost attack** [31] also known as R-U-Dead-Yet (RUDY) attack. The attacker sends HTTP post commands slowly to bring down Web server. The attacker sends a complete HTTP header that defines the ”content-length” field of the post message body as it sends this request for benign traffic. Then it sends the data to fill the message body at a rate of one byte every two minutes as an example as shown in Figure 2.5. Hence, the server waits for each message body to be completed while Slowpost attack grows rapidly which causes the DoS flooding attack on the Web server. The attacker aims to keep the socket opened as long as possible, while he opens many connections to the server to do a DoS attack.

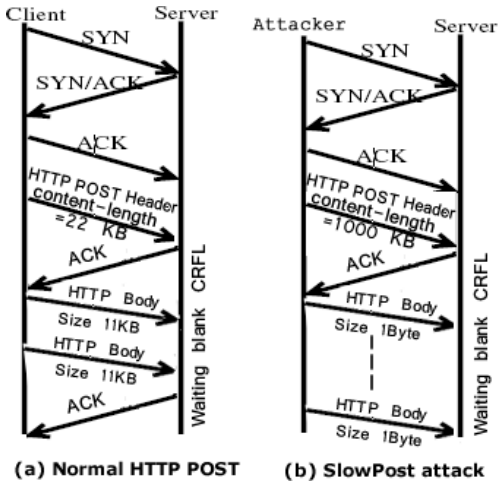


**Figure 2.5 Overview of a normal HTTP POST and a Slowpost attack**

**Slowreading attack** [31] works by slowly reading the response instead of slowly sending the requests. This attack achieves its purpose by setting a smaller receive window-size than the target server's send buffer. The TCP protocol maintains open connections even if there is no data communication as shown in Figure 2.6; hence, the attacker can force the server to keep a large number of connections open and eventually causes the DoS flooding attack on the server. The attacker aims to keep the socket opened as long as possible, while he opens many connections to the server to do a DoS attack.

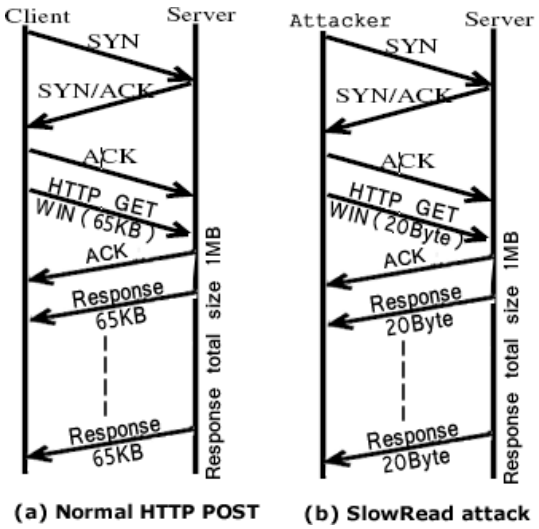


**Figure 2.6 Overview of a normal HTTP read and a SlowRead attack**

### 2.1.3 Overview of DoS and DDoS Attack

**Denial-of-service (DoS) attack** [24] generally consists of efforts to temporarily or indefinitely interrupt or suspend services of a host connected to the internet. Denial-of-service effect is achieved by sending messages to the target machine such that the "message" hampers with its operation and makes it hang, crash, reboot, or do useless work. Also some key resources of the target machine such as bandwidth, CPU time, memory, etc can be consumed by sending a vast number of packets. One cannot attend to legitimate clients because the target application, machine, or network spends all of its critical resources on handling the attack traffic.

**Distributed Denial-of-service (DDoS) attack** [24] Distributed is a special kind of DoS which goal is to increase the attacks intensity by using a number of computers. DDoS attacks are considerably more effective than DoS because they allow to increase the attack intensity by simultaneous use of number of computers. DDoS attempt to make a machine or network resources unavailable to its intended users. It generally consist of the efforts of one or more people to temporarily or indefinitely interrupt or suspend services of a host connected to the internet. DDoS attacks are able to take out an entire server in a matter of minutes. To overwhelm a service to the point where it no longer works is the goal of any DDoS attack.

### 2.1.6 Intrusion Detection Types

Intrusion detection systems types divided mainly based on their scope into two main types, network based (NIDS) and host based (HIDS) intrusion detection systems [32]. Network Intrusion Detection Systems (NIDS) are placed at a strategic network point or points within the network to monitor and analyze the traffic come from or to all devices on the network in order to detect any illegal/abnormal activity. Whereas Host Intrusion Detection Systems (HIDS) run on individual hosts or devices on the network. A HIDS monitors the inbound and outbound packets from the device only and notify the user or administrator if suspicious activity is detected. Our approach is a NIDS.

### 2.1.7 Network Intrusion Detection Techniques:

There are two major techniques of detection in NIDS, signature based and anomaly based. In signature based NIDS, the system looks for the characteristics of known network attacks, stored in its own database, to detect the existence of such attacks, but it fails to detect novel attacks with different characteristics; this failure is known as zero-day attack. Growing number of zero day attacks and the increasing diversity and polymorphism of network attacks made anomaly based NIDS more efficient. By using this way it is possible to detect novel and unknown network attacks without signatures database of known attacks. Today the challenge is to find a way to have fewer false alarms and higher detection rate of complex attacks, especially in imbalance network traffic [33, 34]. Our proposed approach is an anomaly detection technique which based on measuring the deviation of any network instance from the normal behavior of the used service using the standard deviation.

## *2.2 Supervised and Unsupervised Learning*

Data mining algorithms can be organized into two major learning methods [35] which are:

**Supervised Methods:** The main goal of the supervised methods is to build a predictive model (classifier) to classify or label incoming patterns. The classifier has to be trained with labeled patterns to be able to classify new unlabeled patterns. The given labeled training patterns are use to learn the description of classes. Some supervised methods include support vector machines, neural network and genetic algorithms among others [35].

**Unsupervised Methods:** Unsupervised methods take a different approach by grouping unlabeled patterns into clusters based on similarities. Patterns within the same clusters are more similar to each other than they are to patterns belonging to different clusters. Data clustering is very useful when little priori information about the data is available [35].

## *2.3 Data Mining*

It is non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. Also, it is the process of extracting knowledge hidden from large volumes of raw data. The knowledge must be new, not obvious, and must be able to use it. Many people treat data mining as a synonym for another popularly used term, Knowledge Discovery from Data, or KDD. Alternatively, others view data mining as simply an essential step in the process of knowledge discovery [36].

### 2.3.1 Data Preprocessing

Knowledge discovery as a process consists of an iterative sequence of processes starting with data preprocessing which has the following steps [37].

- **Data Cleaning:** is the process of removing incomplete, noisy and inconsistent data.
- **Data Integration:** where multiple data sources may be combined. These sources may include multiple databases, data cubes, or flat files.
- **Data Transformation:** where Data is transformed into forms appropriate for mining by using methods which include Smoothing, aggregation, generalization and normalization.
- **Data Reduction:** where data relevant to the analysis task are retrieved from the database. So, irrelevant, weakly relevant or redundant attributes may be detected and removed the three basic operations in a data-reduction process are delete a column (feature selection), delete a row (sampling) , and reduce the number of values in a column (Discretization).

#### *2.3.1.1 Feature Selection*

The process of choosing features (attributes) that are relevant to a data-mining application in order to achieve maximum performance with the minimum measurement and processing effort [38]. This process includes removing redundant features and irrelevant features which contain no information that is useful for the data mining task.

Finding a good subset of the original attributes is a hard process [38]. Suppose that a dataset contains n attributes, then there are $2^n$ possible subsets. Methods like exhaustive search for the optimal subset of attributes can be too expensive, especially when n and the number of data classes increase. Therefore, heuristic methods that explore a reduced search space are commonly used for attribute subset selection. These methods are typically greedy in that, while searching through attribute space, they always make what looks to be the best choice at the time. Such greedy methods are effective in practice and may come close to estimating an optimal solution. The best attributes are typically determined using tests of statistical significance, which assume that the attributes are independent of one another. Many other attribute evaluation measures can be used such as the information gain measure used in

building decision trees for classification [38]. In our methodology we used the greedy method in order to find the optimal features subset.

**Information Gain** is an attribute selection measure. This measure is based on information theory, Let node N hold the tuples of partition D. The attribute with the highest information gain is chosen as the splitting attribute for node N. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple [38]. The expected information needed to classify a tuple in D is given by equation Eq 2.1

$$Info(D) = -\sum_{i=1}^{n} \mathrm{p}i \log_2(\mathrm{p}i) \quad \text{---------- (Eq. 2.1) [38]}$$

Where pi is the nonzero probability that an arbitrary tuple in D belongs to class Ci.

### *2.3.1.2 Data Normalization*

Normalization or what so-called standardization is a process that attempts to give all attributes an equal weight [38]. Normalization is particularly useful for classification algorithms involving neural networks or distance measurements such as nearest-neighbor classification and clustering [38, 39]. For distance-based methods, normalization helps prevent attributes with initially large ranges (e.g., income) from outweighing attributes with initially smaller ranges (e.g., binary attributes). It is also useful when given no prior knowledge of the data [38, 39].

There are many methods for data normalization. The most common used methods are min-max normalization, z-score normalization, and normalization by decimal scaling. For our discussion, let A be a numeric attribute with n observed values, v1, v2, …, vn.

**Min-Max Normalization** performs a linear transformation on the original data. Suppose that minA and maxA are the minimum and maximum values of an attribute, A. Min-max normalization maps a value, vi, of A to v′i in the range [new_minA, new_maxA] by computing equation Eq 2.2, note that min-max normalization will encounter an "out-of-bounds" error if a future input case for normalization falls outside of the original data range for A.

$$\mathrm{v}i' = \frac{\mathrm{v}i - minA}{\mathrm{maxA} - \mathrm{minA}}(new\_maxA - new\_minA) + new\_minA \quad \text{---------- (Eq. 2.2) [38]}$$

**Z-score Normalization** (or zero-mean normalization), the values for an attribute, A, are normalized based on the mean (i.e., average) and standard deviation of A. A value, v*i*, of A is normalized to v*i*′ by computing equation Eq 2.3, where $\mu A$ and $\sigma A$ are the mean and standard deviation, respectively, of attribute A. This method of normalization is useful when the actual minimum and maximum of attribute A are unknown, or when there are outliers that dominate

the min-max normalization. In our model we used this normalization method because the network feature values is unpredictable, also the existence of extreme values may occurs.

$$vi' = \frac{vi - \mu A}{\sigma A}$$ ---------- (Eq. 2.3) [38]

**Normalization by Decimal Scaling** normalizes by moving the decimal point of values of attribute A. The number of decimal points moved depends on the maximum absolute value of A. A value, v*i*, of A is normalized to v*i*′ by computing equation Eq 2.4 where j is the smallest integer such that max(|v*i*′|) < 1. Note that this normalization method will encounter an "out-of-bounds" error if a future input case for normalization greater than the original maximum value of A.

$$vi' = \frac{vi}{10^{j}}$$ ---------- (Eq. 2.4) [38]

Note that normalization can change the original data quite a bit, especially when using z-score normalization or decimal scaling. It is also necessary to save the normalization parameters (e.g., the mean and standard deviation if using z-score normalization) so that future data can be normalized in a uniform manner.

Suarez-Alvarez, M.M., et al. [39] proposed a unified statistical approach to normalization of all attributes of mixed databases, when different metrics are used for numerical and categorical data. It is shown that the classic z-score standardization and the min–max normalization are particular cases of the statistical normalization, when the objective function is based on the Euclidean or the Tchebycheff (Chebyshev) metrics respectively. In their research they proposed and equation based on normalized Euclidean distance for both numerical and categorical features. In our model we apply Euclidean distance, shown in equation Eq 2.5, on z-score normalized categorical and numerical features.

$$d(i,j) = \sqrt[2]{\sum_{f=1}^{F}(x1f - x2f)^2}$$ ---------- (Eq. 2.5) [38]

## *2.4 OCC Methodologies*

Khan and Madden [14] classify OCC methodologies used into two main categories**,** (1) Methods based on One Class Support Vector Machine (OCSVM) and (2) methods based on non-OCSVM. The methods that are based on OCSVM are based mainly on Gaussian kernel function to build the model and determine the class boundaries. Beside the advantages of SVMs, they have an important practical problem that is not entirely solved, which is the selection of the kernel function parameters - for Gaussian kernels the width parameter σ, these parameters are not obtained from the dataset [40, 41].

The non-OCSVM methods are based on different techniques, some are based on Gaussian kernel function, others based on decision trees [42]. The shortcomings of methods based on

decision trees that they required the existence of three sets of examples as an input to construct the decision trees which are :(1) as set of labeled examples, (2) a set of positive examples and (3) unlabeled examples.

We proposed a novel method for building OCC based on the standard deviation in our previous work [16]. OCC based on standard deviation is illustrated in Figure 1.2, as shown the normal boundary from the class center is the first circle which is the standard deviation of it and any expanding of this boundary will increase the false alarm and decrease the intrusion detection rate based on the approach proposed by us in [16]. As shown in Figure 1.2, we need to adjust the class boundary in order to achieve low false alarm rate. We use the term "Tune" in our work which means an positive real value added to the normal class's standard deviation.

## *2.5 NID Using Data mining:*

Data mining classification and clustering techniques have been used in anomaly based NIDS and improve the performance of attack detection [43]. There are three categories of data mining classification and clustering techniques for NIDS which are supervised; semi-supervised and unsupervised learning techniques [16, 43, 44] in addition to OCC [14] which is a special case of unsupervised learning techniques. Supervised learning technique needs to be trained firstly by pre-classified traffic sample to build the classification model and map the behavior of the network to find the difference between normal and abnormal state. The shortcomings of this technique is that the system is trained on the existing attacks, which may fail to detect a novel variant attacks [45], also in most circumstances, labeled data is not readily available since it is time consuming and expensive to manually classify it [46-48]. In many practical applications there are a massive data which are often unlabeled like mail spam. The limited labeled data are not enough to train a supervised classifier with fine generalization performance.[49].

Many researchers have tried to address these problems by using unsupervised learning techniques such as clustering [47, 48, 50]; by using clustering techniques, they try to measure the deviation of the new instances from the different created clusters. Clustering is the process of assigning a set of objects into group or groups (which called cluster) while the objects in the same cluster are more similar (in some way) compare to other objects [51]. But labeling these clusters is a great problem; which cluster should be labeled as normal and which should be labeled as abnormal [52]. Laskov, Düssel et al. [45] carried out an experimental framework for comparative study of various supervised and unsupervised approaches for intrusion detection. Their results indicate that the problem of unlabeled data being drawn from a different distribution remain unsolved within the purely supervised or unsupervised techniques and they put their marks on semi-supervised learning approaches that it may provide the superior intrusion detection ability.

To overcome the shortcomings of supervised and unsupervised learning techniques, especially in applications like intrusion detection systems which manipulate and analyze a

huge number of different packets, a third learning strategy which called semi-supervised learning is being used [53]. This technique exploits unlabeled data in addition to labeled ones. Many researchers have used this technique in intrusion detection [16, 49, 54]. Although this learning technique solved the problem of labeling instances and gain the ability of prediction based on relatively a few labeled examples, it suffers from the limitation that it cannot outperform supervised classification unless the analyst is absolutely certain that there is some nontrivial relationship between labeled and the unlabeled distribution [55]. It is also well known that the utilization of unlabeled dataset U is not always helpful for semi-supervised learning algorithms. In particular, it is not guaranteed that adding U to the training data, T, which has a labeled instances L i.e., $T = L \cup U$, leads to a situation in which we can improve the classification performance [3, 55]. When Semi-Supervised learning assumptions are made, but do not hold, it can degrade the performance and can be worse than supervised learning [55]. In addition, semi-supervised learning consumes time in labeling process, e.g. Self-training algorithm which need more than one iteration to label unlabeled dataset, another algorithm known as Co-training which depends on the existing of two classifiers built on different features subsets extracted from the main features set [53]. Besides that the classifier model is learned on certain attacks, but it may fail to detect novel variant attacks based on the classifier used [45].

Because of the previous mentioned detection techniques limitations and shortcomings, and because of the increased diversity of attacks that we can't predict its future behavior, an alternative detection technique that can overcome these obstacles is needed. So, we need a learning technique that learns just the normal behavior and detect any deviation from it. This technique is known as One-Class Classification (OCC). Because of the increasing diversity and polymorphism of network attacks which means that very few of these attacks are known, or they do not form a statistically-representative sample of the negative concept. So there's an urgent need to learn how the positive class behave to detect any deviation from it which may be a negative class.

Many algorithms for intrusion detection based on OCC have been propose [16, 56-59], many of them have used One Class Support Vector Machines (OCSVMs) which is based on Gaussian Kernel function. Others have used other techniques such as *v*-SVC [59] and standard deviation [16]. Most of the proposed NIDSs that have applied OCC deal with the whole network instances as a single class, so their proposed NIDs suffer from the high dimensional network feature spaces, and also from the existence of large differences in density which affect the detection accuracy. As far as we know almost all of them have not considered to detect attacks based on the standard deviation of normal behavior of the used service such as HTTP service.

To overcome these challenging issues in OCC, we will propose a primary OCC-NIDS model based on the standard deviation of network service's normal behavior. Through this model we deal with each network service as single class instead of dealing with all network services as a single class. By this way we just use the relevant features of each service, hence reducing the high dimensional network feature spaces and also ensure that each class has - a proximately -

uniform distribution.

Based on the learning techniques which have been mentioned above, little attention has been given towards the separation of relevant features from the overall dataset based on the service used. Each service has its own feature space and its own characteristics and behavior, and as far as we know after deep searching and digging, no one have used the standard deviation in detecting the deviation of new network instances from its same service's normal behavior. In our recent research [16], we used the standard deviation in detecting the deviation of new network instances from its same transport protocol's normal behavior. But we face a problem of large differences in density within single transport protocol class, which limited us from detecting some attacks. Also the feature space of one class was high because of the existence of all services features, and this also affected the distance measurements because we calculated the distance between a new service instance with irrelevant features that belong to other service. (e.g. feature to tell if SMTP instance initiate communication with HELO is irrelevant to other services).

Due to the explosive growth of the network attacks, intrusion detection systems have become an essential network component which plays a vital role for computer networks' security. A lot of efforts have been given toward designing a perfect NIDS that has a high detection rate and low false alarm rate. Some have used misuse detection technique which fails to detect zero-day attacks, such that there is a high demand for alternative detection techniques. Many researchers are trying to solve the problem by using data mining classification/clustering techniques; the problem of using supervised learning techniques is the cost of producing labeled dataset which is essential for training the model and also the model is trained on a known attacks which may fail to detect new variant attacks. On the other hand, unsupervised learning has the problem of labeling the generated clusters. Semi-supervised learning techniques suffers from the limitation that it cannot outperform supervised classification unless the analyst is absolutely certain that there is some nontrivial relationship between labels and the unlabeled distribution. Because of the limitations of previous techniques, and because of the increasing diversity and polymorphism of network attacks, a fourth learning technique called OCC has been used. However this technique suffers from the high dimensional network feature spaces. Also, problems may arise when large differences in density exist. To overcome these problems, we will propose a primary OCC-NIDS model based on the standard deviation of service's normal behavior. Through this model we deal with each network service as single class instead of dealing with all network services as a single class. By this way we use just the relevant features of each service protocol, hence reducing the high dimensional network feature spaces and also ensure that each class has - a proximately - uniform distribution.

## *2.6 KDD Cup'99 Dataset*

KDD Cup'99 is a benchmark used in NIDS researches. We chose it in order to evaluate our model. KDD Cup'99 [20] dataset was prepared and managed by MIT Lincoln Labs. Lincoln Labs sat up an environment to acquire nine weeks of raw TCP dump data for a local-area

network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment, but peppered it with multiple attacks. The simulated network represents thousands of UNIX hosts and hundreds of users. There are three UNIX machines designated as victim machines running three different operating systems: SunOS, Solaris OS, and Linux [23]. Figure 2.8 shows the test bed block diagram

The training data was processed to about five million connections records from seven weeks of network traffic and two weeks of testing data yielded around two million connection records. The training data is made up of 22 different attacks out of the 39 present in the test data. The known attack types are those present in the training dataset while the novel attacks are the additional attacks in the test datasets not available in the training data sets [20]. The attacks types are grouped into four categories [23] which are: DoS attacks, Probing attacks, U2R attacks, and R2L attacks.

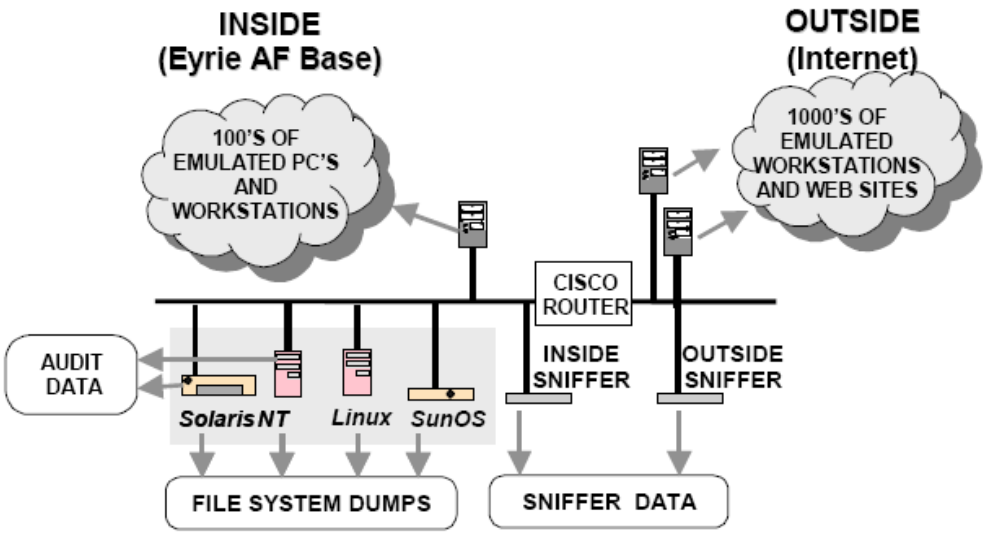


**Figure 2.8 Block diagram of KDD Cup'99 test bed [23]**

The training dataset consists of 494,021 records among which 97,277 (19.69%) were normal, 391,458 (79.24%) DOS, 4,107 (0.83%) Probe, 1,126 (0.23%) R2L and 52 (0.01%) U2R connections as shown in Table 2.1. In each connection there are 41 attributes describing different features of the connection and a label assigned to each either as an attack type or as normal.

**Table 2.1 KDD Cup'99 Normal and Abnormal instances distribution of training dataset**

| KDD dataset | Normal | Abnormal | | | | Total |
|---|---|---|---|---|---|---|
| | | DoS | Probe | R2L | U2R | |
| 10% | 97277 | 391458 | 4107 | 1126 | 52 | |
| ∑ | 97277 | 396743 | | | | 494020 |

We need to use only the normal data, extracted from 10% training dataset, to build our model and the model evaluation will taken place using the 10% testing dataset which means that all attacks are new to our model because our model didn't trained on them , it's only trained on the normal instances.

It is important to note that the test data is not from the same probability distribution as the straining data, and it includes specific attack types not in the training data. This makes the task more realistic. Some intrusion experts believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants. The datasets contain a total of 24 training attack types, with an additional 14 types in the test data only.

We have used three services to build their OCC from KDD Cup'99 dataset, these services have enough normal instances and attack instances. These services are HTTP, ECR_I and POP3 services.

**HTTP Service**

Hypertext Transfer Protocol (HTTP) [RFC 2616] is an application layer protocol which is the most common used protocol on the internet. Its default TCP Port is 80 and it's based on TCP protocol which is an network layer protocol. HTTP is used for distributed, collaborative, hypermedia information systems [60], HTTP is the foundation of data communication for the World Wide Web .

**POP3 Service**

Post Office Protocol 3 (POP3) [RFC 1939] is an application-layer Internet standard protocol with a default TCP Port 110, used by local e-mail clients to retrieve e-mail from a remote server over a TCP/IP connection [61]. E-mail clients using POP3 generally connect, retrieve all messages, store them on the user's PC as new messages, delete them from the server, and then disconnect.

**ECR_I Service**

ECR_I [RFC 792] is an Echo-Replay service in ICMP protocol that have an ICMP code 0. This service is a replay for an Echo-Request message sent by the requester using ICMP code 8 [RFC 792]. ICMP Echos are used mostly for troubleshooting. When there are 2 hosts which have communication problems, a few simple ICMP Echo requests will show if the 2 hosts have their TCP/IP stacks configured correctly and if there are any problems with the routes packets are taking in order to get to the other side. The Internet Control Message Protocol (ICMP) is one of the main protocols of the Internet Protocol Suite. It is used by network devices, like routers, to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached [62]. ICMP [63] differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications with the exception of some diagnostic tools like ping and traceroute. Attacks that exploited the three services are listed in Table 2.2 [64]

**Table 2.2 Attacks exploited HTP, POP3, and ECR_I services in KDD Cup'99 dataset**

| Type | Attack | Service exploited | Describtion |
|---|---|---|---|
| Prope | saint.,satan. | HTTP, ECR_I | it gathers information about remote hosts and networks. These flaws include incorrectly setup or configured network services, well-known bugs in system or network utilities, and poor policy decisions. |
| | ipsweep. | HTTP, ECR_I | Is a surveillance sweep to determine which hosts are listening on a network. |
| | portsweep. | HTTP, ECR_I, POP3 | is a surveillance sweep to determine which ports hosts are listening on a network. |
| | nmap. | ECR_I, POP3 | The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order. |
| | mscan. | POP3 | Is a probing tool that uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and test them for vulnerabilities. |
| DoS | neptune. | HTTP, POP3 | A SYN Flood is a denial of service attack to which every TCP/IP implementation is vulnerable. |
| | back. | HTTP | DoS attack against the Apache web server, an attacker submits requests with URL's containing many front-slashes. As the server tries to process these requests it will slow down and becomes unable to process other requests. |
| | apache2. | HTTP | DoS attack against an apache web server where a client sends a request with many http headers. If the server receives many of these requests it will slow down, and may eventually crash. |
| | smurf. | ECR_I | Attacker sends ICMP echo request packets to the broadcast address xxx.xxx.xxx.255 of many subnets with the source address spoofed to be that of the intended victim. Any machines that are listening on these subnets will respond by sending ICMP 'echo reply' packets to the victim. The smurf attack is effective because the attacker is able to use broadcast addresses to amplify what would otherwise be a rather innocuous ping flood. |
| | pod. | ECR_I | The Ping of Death is a denial of service attack that affects many older operating systems. Although the adverse effects of a Ping of Death could not be duplicated on any victim systems used in the 1998 DARPA evaluation, it has been widely reported that some systems will react in an unpredictable fashion when receiving oversized IP packets. Possible reactions include crashing, freezing, and rebooting. |
| R2L | phf. | HTTP | The Phf attack abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function escape_shell_cmd() to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server. |
| | guess_pass. | POP3 | In this attack, the attacker try to guess POP3 account password by sending multiple login quires. |

## *2.7 Summary*

In this chapter, intrusion definition, types, characteristics, intrusion detection techniques, supervised and unsupervised learning definitions and data mining classification/clustering techniques were identified. In addition, main problems for current data mining classification /clustering learning approaches were discussed. These problems are: labeling network traffic dataset to be used with supervised learning, labeling generated clusters to normal or abnormal traffic, and also the existence of high dimensional network feature spaces. OCC was discussed and the detection technique using it was explained. KDD Cup'99 dataset was described and the services that were included in our model evaluation were presented and listed the attacks exploited them with a brief description.

# Chapter 3: Related Works

Data mining's classification and clustering techniques have been widely used in NIDS and improve the performance of attack detection. Three categories of data mining's classification and clustering techniques have been used for NIDS which are supervised, semi-supervised and unsupervised learning techniques in addition to OCC learning technique which is a special case of clustering techniques. In the following subsections we introduce some of these researches which related to our research.

## *3.1 Supervised NIDS*

**Chandolikar and Nandavadekar [65]** proposed a model based on feature selection and rule induction on a KDD Cup' 99 dataset [20] to get appropriate rules for intrusion detection. Their approach is implemented to detect 5 different classes of attacks from the dataset including DoS, U2R, Probe, R2L and normal. The experiment show high overall detection rate 96.18%, but the detection rate of R2L attacks and U2R attacks is 79% and 0% respectively which are unacceptable detection rates for such attacks which are more dangerous than DoS attack because of integrity and confidentiality violation. The shortcomings of their model is the need of efficient representative data of all possible attacks that may exploit the system in the future, but it's impossible to have a representative dataset, this means that their model may fail to detect novel attacks with different behavior and distribution. On the other hand it's hard, time consuming and expensive to manually classify the dataset in real environment. Moreover, the model suffers from high dimensional feature space because of the existence of all relevant network features.

**Almutairi and Parish [66]** proposed a predictive intrusion detection model that is based on usage of classification techniques such as decision tree and Bayesian techniques. The model was trained using KDD'99 [20] intrusion detection dataset. The results showed that decision tree algorithm J48 based on C4.5 provides 99.95% of correctly classified instances and was better than the Naïve Bayes technique. Also found that false positives using Naïve Bayes was high for Probing and Remote to Local attack categories. The shortcoming of their model is the inability to detect novel attacks with different distribution than those attacks in the training dataset based on classifier used. The feature selection also may eliminate some features which are relevant to future attacks. Also it's a problem to have labeled data in real environment.

**Xie et al. [67]** proposed a new network intrusion detection classification model based on the Support vector machine (SVM). Their model used the factor analysis algorithm to convert a large number of related features into concise integrated features, and the support vector decision function ranking method calculated the contribution of network behavior features. Then some important network features were extracted and network behaviors were classified consequently. The experimental results showed that the detection rate and the real-time of this classification model are satisfying 99.03%-99.7% . The shortcomings of their model is the need of labeled dataset in order to build the classifier system which is hard to be obtained in

real environment and also their model may fail to detect novel attacks with different behavior and distribution. Moreover, the model suffers from the existence of all relevant network features which consume time to classify a new instance. Also the time consumed to classify an instance is large because of using high number of support vectors.

**Su [68]** proposed a method to identify flooding attacks in real-time, based on anomaly detection by genetic weighted KNN (K-nearest-neighbor) classifiers. A genetic algorithm is used to train an optimal weight vector for features; mean while, an unsupervised clustering algorithm is applied to reduce the number of instances in the sampling dataset, in order to shorten training and execution time, as well as to promote the systems over all accuracy. More precisely, instances in the sampling dataset are replaced by less, but more significant, centroids of clusters. According to the proposed method, the system is implemented and evaluated by numerous Denial-of-Service (DoS) attacks. With an embedded weighted KNN classifier, the proposed system could identify a DoS attack from network traffic within a very short time. The experimental results show that the proposed system could achieve 95.86% in overall accuracy in the case of 2 -fold cross- validation, and 96.25% in overall accuracy for all known attack evaluations, The problem in this model is that they need to determine the number of clusters (K) manually. Also the proposed system is not suitable to detect attacks other than flooding attacks; it may not detect also some DoS attacks that exploit a system bug (e.g. sending malformed packet which break down the system by just one packet). Also, the existence of all network features consumes time when measuring the distance between a new instance and the other clusters which is a critical issue in NIDS's .

**Dartigue et al. [69]** proposed a new data-mining based technique for intrusion detection using an ensemble of binary classifiers with feature selection and multiboosting simultaneously. Each attack type (DoS, Prope, R2L and U2R) has a binary C4.5 classifier in addition to Normal binary C4.5 classifier, and each classifier has its feature subset selected. Based on the accurate binary classifiers, their model applied a new ensemble approach which aggregates each binary classifier's decisions for the same input and decides which class is most suitable for a given input. During this process, the potential bias of certain binary classifier could be alleviated by other binary classifiers' decision. Their model also makes use of multi-boosting for reducing both variance and bias. The experimental results on KDD Cup'99 dataset [20] show that their approach has low false alarm rate but high false positive rate specially in R2L and U2R attacks which are the most dangerous attacks that exploit the CIA components, confidentiality and integrity. The overall accuracy of their model is 92.3%. The shortcomings of their approach is the cost to classify a new instance which need to be passed through the 5 binary classifiers to make decision about its class, in addition, R2L and U2R attacks are service dependent attacks which means that their features set varies based on the service that they exploit.

**Abd-Eldayem [70]** proposed a new HTTP-IDS based on Naïve Bayes classifier. In the training phase of the proposed IDS, at first a feature selection technique based on Naïve Bayes classifier is used, which used to identify the most important HTTP traffic features that can be used to detect HTTP attacks. In the testing and running phases, the proposed IDS

classifies the network traffic based on the requested service, then based on the selected features, Naïve Bayes classifier is used to analyze the HTTP service based traffic and identifies the HTTP normal connections and attacks. The performance of the IDS is measured through experiments using KDD Cup'99 dataset [20]. The results show that the detection rate of the IDS is about 99%, the false-positive rate is about 5%, and the false-negative rate is about 0.6%. The shortcomings of their model is the need of efficient representative data of all possible attacks that may exploit the system in the future, but it's impossible to have a representative dataset, this means that their model may fail to detect novel attacks with different behavior and distribution. On the other hand it's hard, time consuming and expensive to manually classify the dataset in real environment.

## *3.2 Unsupervised NIDS*

**Jiang et al. [50]** proposed a new strategy for intrusion detection. It consists of three stages, based on clustering training data, then sort clusters according to their outlier factor. Then label some clusters that contain percentage e of the data as 'normal' while labeling the rest of the clusters as 'attack'. They regard labeled clusters as model, and detect an object whether it is an attack or not by the distance between an object and the nearest cluster. They considered the outlier factor of clusters for measuring the deviation degree of a cluster. A novel method has been proposed to compute the cluster radius threshold. The data classification has been performed by an improved nearest neighbor method. The experiments demonstrated that their method outperforms the existing methods in terms of accuracy and detecting unknown intrusions. Their proposed method is effective when almost data in the network is a normal, but this assumption fails in flooding attacks which outnumbers the normal traffic. Also the number of generated clusters is high which means that it consumes time to classify a new instance.

**Amoli and Hamalainen [48]** proposed a new Real Time Unsupervised NIDS which monitor network flows in two windows with different sizes and detect network attacks by correlating outliers from multiple clusters. The proposed NIDS has the ability of detecting different types of intrusions in real-time such as DOS, DDOS, scanning, distribution of worms and any other network attacks which produce huge amount of network traffic and in the meanwhile it detects Bot-Master if the detected attack lunched by Bots. The authors didn't mentioned how to distinguish between normal and abnormal packets, also the limitation of this approach is the use of DBSCAN algorithm which fails when the density is vary in normal instances. Also it can't detect attacks other than flood attacks. Moreover, the model suffers from high dimensional feature space because of the existence of all relevant network features.

**Bhuyan et al. [44]** proposed an unsupervised IDS using a tree based subspace clustering technique for generating clusters in high dimensional large datasets. Their approach exploits a specific technique for finding a highly relevant feature set. The clustering technique used based on the stability of the obtained cluster. Their approach decrease false alarm, while increase the percent of detection rate reaches between 89.3% - 99.1%. The problem in this IDS is that the stability of cluster is not exclusive in normal clusters, but also in abnormal

clusters such as DoS. In addition, they didn't determine the techniques that have been used for choosing relative features.

**Hameed and Sulaiman [46]** proposed an algorithm for intrusion detection that combines both fuzzy C Means (FCM) and FCM for symbolic features algorithms in order to manipulate with network traffic data stream that contains symbolic features in addition to the numeric features. KDD Cup'99 dataset [20] was used to evaluate their model. Experimental results show that the average detection rate of the proposed algorithm was 99%. In this paper, the authors consider the distance measurement of a dataset contains numerical and symbolic data types, but they didn't mention how they label the resulted clusters. Moreover, the proposed algorithm suffers from high dimensional feature space because of the existence of all relevant network features.

**Leung and Leckie [47]** proposed a density based and grid based clustering algorithm, named as fpMAFIA, that uses adaptive grid algorithm adopted from pMAFIA and FP-tree growth method for frequent item set mining. They aim to discover clusters from large volume of high dimensional input data. Grid-based methods divide the object space into a finite number of cells that form a grid structure. All of the clustering operations are performed on the grid structure. Once they obtain the set of clusters, they expect that they cover most but not all of the data set. Therefore any point that falls inside the clusters will be labeled as normal. The small percentages of points that do not belong to any clusters are labeled as abnormal. Their solution has the advantage that it can produce clusters of any arbitrary shapes and cover over 95% of the data set with appropriate values of parameters. They have evaluated the accuracy of the new approach and show that it achieves a reasonable detection rate 97.3% while maintaining a low positive rate. The problem of their work that they consider the large cluster as normal, but DoS attacks has also a large number of similar instances. In addition, they assume a small percentage of points that do not belong with any clusters are labeled as abnormal, but in the real network this is not always true. Another problem is the consuming time for extracting frequent item sets from high dimensional feature space.

**Han [71]** proposed an KMIE-IDS based on K-MEANS and information entropy to detect anomaly activities. His aim was to improve the detection rate and decrease the false alarm rate. KMIE can filter the outliers on the dataset to reduce the negative impact, and indentify the initial cluster centers using entropy method. Then, KMIE can use these centers to iterative calculate and classify records into different clusters. They used KDD Cup'99 dataset [20] to test the performance of KMIE algorithm. The results show that their method has a high detection rate 95.9%, and a relatively high false alarm rate 2.4%. The test was done on different attacks came from four different classes (a DoS attack, a probe attack, an R2L attack, and an U2R attack). The testing results show high false positive rate specially in R2L and U2R attacks which are the most dangerous attacks that exploit the CIA components, confidentiality and integrity. The author didn't mention how the labeling technique works. Moreover, the proposed algorithm include all network features which consumes time for labeling a new instance.

### 3.2.1 One-Class Classification

**Barhoom and Matar [16]** proposed a novel OCC learning technique based on the standard deviation of transport protocol's normal behavior. The transport protocol are TCP, UDP and ICMP. By this technique they measured the deviation of any new instance from the same transport protocol class. The standard deviation of each transport protocol class is being the class radius, if the distance between the new instances and the relative transport protocol's class greater than the class standard deviation then the instance is labeled as abnormal else it is labeled as normal. The experimental results on KDD Cup'99 dataset [20] show high detection rate 87.7%-99.2% with low false alarm 1.16%. This work suffers from the high dimensional feature spaces of each transport protocol's class and also each transport protocol's class has varying density. These problems arise due to the existence of several network services in each class (e.g. HTTP, SMTP in TCP class) which affects the overall detection rate and false alarm rate.

**Araki et al. [57]** proposed a multistage intrusion detection model based on OCSVM focusing on communication interval. The multistage OCSVM uses three sets of traffic, two sets retrieved from a traffic archive and one extracted from real network. At the first stage, OCSVM learns older archive set and then analyzes newer archive set and one from real network. At the second stage, OCSVM learns outlier traffic from the newer archive set and analyzes that from the real network. As a result, extracted traffic from outlier of the real network which does not exist in the newer set can be extracted. They evaluated their method using Kyoto2006+ [72] Dataset and 6 new features. The results show that their method detects attacks with 94% detection rate and 6% false positive rate. The proposed algorithm suffers from high dimensional feature spaces. The increase of feature space is due to the existence of all network features which affect the detection rate, because of measuring the distance between irrelevant service-based features.

**Winter et al. [58]** proposed inductive network intrusion detection system. The system operates on lightweight network flows and uses One-Class Support Vector Machines for analysis. But the system was trained with malicious rather than with normal network data. Evaluations brought satisfying results. They achieved 0% false alarm with detection rate around 98%. The drawbacks of this work that the attack variations are unlimited, this leads to have big differences in class density which affect the detection performance of the OCSVM. Also it is impossible to have a representative dataset of all possible attacks that could happen in the future.

**Giacinto et al. [59]** proposed an unlabeled Network Anomaly IDS based on a modular Multiple Classifier System (MCS). Each module is designed to model a particular group of similar protocols or network services. The use of a modular MCS allows the designer to choose a different model and decision threshold for different (groups of) network services. This also allows the designer to tune the false alarm rate and detection rate produced by each module to optimize the overall performance of the ensemble. Experimental results on the KDD Cup'99 dataset [20] show that the proposed anomaly IDS achieves high attack detection rate and low false alarm rate at the same time. They achieve detection rate around 94% with false alarm around 9%. Their work is similar to ours but differs in the technique used. They

use *v*-SVM to build their OCC model. Beside the advantages of SVMs, they have an important practical problem that is not entirely solved, which is the selection of the kernel function parameters - for Gaussian kernels the width parameter σ [40, 41].

**Ma and Dai [73]** proposed anomaly detection using dissimilarity-based one-class classifiers (DBOCCs) with unsupervised learning approach. Several combinations of DBOCCs scheme have also been used. This technique is proposed in order to solve the drawback of traditional features-based classifiers which suffer from the improper features selection. The dissimilarity based OCCs are constructed on dissimilarity representations (DR). The experimental results on KDD Cup'99 [20] dataset show that DBOCCs can achieve high detection rate and low false positive rate without large degeneration in performance, as traditional feature-based classifiers suffered when different feature subsets have been used. They achieve 95% detection rate with individual OCC, and around 98% with combined OCC. They didn't show the false alarm rate. The proposed algorithm suffers from high dimensional feature spaces. The increase of feature space is due to the existence of all network features which affect the detection rate, because of measuring the distance between irrelevant service-based features.

**Zainal et al. [74]** proposed an ensemble of one-class classifiers where each adopts different learning paradigms. The techniques deployed in this ensemble model were; Linear Genetic Programming (LGP), Adaptive Neural Fuzzy Inference System (ANFIS) and Random Forest (RF). The strengths from the individual models were evaluated and ensemble rule was formulated. Prior to classification, a 2-tier feature selection process was performed to expedite the detection process. The feature set is selected for each attack type, DoS, Prope, R2L and U2R, in addition to the normal class, and the output is one of the five classes. Empirical results on KDD Cup'99 dataset [20] show an improvement in detection accuracy for all classes of network traffic; except DoS and U2R with 97.43% and 88% respectively. The overall accuracy of their model is 96.57%. The shortcomings of their approach is the cost of classifying a new instance which need to be passed through the three OCC to make decision about its class, in addition, U2R attacks are service dependent attacks which means that their features set varies based on the service that they exploit.

## *3.3 Semi-supervised NIDS*

**Chen et al. [75]** proposed three semi-supervised approaches, from these two are classification methods based on (Graph Transducer and Gaussian Fields) and one is clustering method based on (MPCK-means). Classification semi-supervised methods are used to detect unknown Attacks. Clustering semi-supervised method is used to improve the performances of the traditional Purely unsupervised clustering methods. Their experimental analysis show that the performances of proposed semi supervised classification methods are superior than those of the other supervised learning methods for detection of unknown attacks. The shortcomings of the Graph Transducer algorithm is the use of Mincut technique, which has a problem that it only gives hard classification without confidence, which means that it may add a misclassified instance to the training set which affect the final model [53]. Also there is a need of the availability of efficient representative labeled dataset contains both attacks and normal

instances which may not be available. Also the model may fail to classify new variant attacks that have different distribution, because the model is trained on a specific type of attacks [45]. Moreover, all the network features are included in the distance measurement in order to classify a new instance which consumes time.

**Li et al. [49]** proposed a new semi-supervised SVM algorithm using Tri-training algorithm which is an improved version from the standard co-training algorithm, they applied tri-training to improve SVM. The tri-training algorithm consists of three classifiers. The semi-supervised SVM makes use of large number of unlabeled data to modify the classifiers iteratively. Although tri-training doesn't put any constraints on the classifier, the proposed method uses three different SVMs as the classification algorithm. Experiments on UCI datasets and application to the intrusion anomaly detection show that tri-training can improve the classification accuracy of SVM. The shortcomings of this approach is the need of the availability efficient representative of labeled dataset contains both attacks and normal instances which may not be available, and also the model may fail to classify new variant of attacks, because the model is trained on a specific type of attacks [45]. The second issue is the time consumed by the three classifiers in order to classify the unlabeled instance in more than one iteration to increase the labeling confidence. Moreover, the existence of all network features increase the time needed to classify a new instance.

**Wagh and Kolhe [76]** proposed intrusion detection approach using self-learning algorithm (SLA). In this algorithm the labeled data are used for training and unlabeled data is used for testing. Then the most confident data – based on threshold- with predicted labels from the output of the testing phase is selected and added in the labeled data. The learned set formulation helps to remove the data redundancy in the labeled data and controlled the size of the labeled data. All experiments are carried out with KDD Cup'99 dataset [20]. Accuracy of intrusion detection in first iteration is 97.286%, for second iteration it is 99.511% and for third iteration it is 99.516%. The final accuracy for DoS, U2R, R2L and Probe is 99.25%, 66.66%, 70%, and 96.88% respectively. The shortcomings of this technique is that the training data must have some instances labeled as attack beside the normal instances for building the initial model which may not be available and also the model may fail to classify new variant of attacks, based on the classifier used, because the model is trained on a specific type of attacks [45]. Another disadvantage is the dependency on the threshold value which may affect the overall result because of adding a misclassified instances in the training dataset in the subsequent iterations. Beside the problem of time consuming which is consumed by multi iteration to obtain the final result. Also including all network features space increase the time neede for classifying a new instance.

**Yang et al. [77]** used both classification and clustering algorithms for intrusion detection. They divided the dataset into different feature vectors according to the service type. They used the standard k-means algorithm for clustering. Each generated cluster can be simply expressed as a centroid and an effect influence radius. The cluster radius refers to influence range of a data point (represented as the Euclidean distance from the centroid), the classification is done by measuring the distance between the new instance with every instance

in the dataset, if the distance fall in within the Gaussian distribution then increment the normal counter *f*+ if the training instance is normal else increment the attack counter *f*-, then the classification is done by checking if $f+/(f- + f+) >$ normal threshold then the new instance is normal, else if $f-/(f- + f+) >$ attack threshold then the instance is labeled as attack else it is labeled as anomaly. There model achieve high detection rate 10%-99.5% and low false alarm 0.7% on KDD Cup'99 dataset [20]. The testing results show high false positive rate specially in R2L (10.44% detection rate) and U2R (81.14% detection rate) attacks which are the most dangerous attacks that exploit the CIA components, confidentiality and integrity. There's some shortcomings in this approach. Firstly, there's a lot of parameters which affect the overall results which are, the number of clusters for each service, the normal threshold value and the attack threshold value. Secondly, the model consumes time in measuring the distance between the new instance and every instance in the dataset, which affect the performance of IDS.

**Li et al. [78]** proposed an intrusion detection algorithm based on semi-supervised fuzzy clustering. For training the model, they used a few labeled samples and many unlabeled samples as seeds initializing the classifier of the system. Under the constraint of labeled data, they used fuzzy C-Means to create clusters. Then used labeled instances for labeling clusters, after that they used the unlabeled instances for improving the clusters labeling. Comparing with FCM algorithm, the experiment results on data sets KDD Cup'99 dataset [20] show the effectiveness of the proposed algorithm, it has higher detection rate 85.50%, but in general their accuracy was relatively lower than other semi-supervised learning approaches.

**Mahajan and Verma [79]** proposed a semi supervised machine learning technique. In their system a distance based semi-supervised clustering and probabilistic assignment technique is used to analyze and implement a network traffic classifier using both labeled and unlabelled flows. They used K-means for clustering and probabilistic assignment for labeling the generated clusters. The available labeled instances are used to obtain a mapping from the clusters to the different known classes. They tested their model using KDD Cup'99 dataset [20] and achieved good classification accuracy up 94.7%. The problem in their model, that they didn't determine how to achieve the optimal number of clusters to be generated. Moreover, the proposed algorithm suffers from high dimensional feature space because of the existence of all relevant network features.

## *3.5 Summary*

A lot of efforts have been given toward designing a perfect NIDS that has a high detection rate and low false alarm rate. Many researchers are trying to solve the problem by using supervised learning techniques. The problem of using supervised learning is the cost of producing labeled dataset which is essential for training the model [46-48] and also the model is trained on known attacks which may fail to detect new variant attacks [45]. Others on other hand are trying to use unsupervised learning techniques which has the problem of labeling the generated clusters [52]; which cluster is to be normal or abnormal. Semi-supervised learning

techniques suffers from the limitation that it cannot outperform supervised classification unless the analyst is absolutely certain that there is some nontrivial relationship between labeled and the unlabeled distribution [55]. Because of the limitations of previous techniques, and because of the increasing diversity and polymorphism of network attacks, a fourth learning technique called One-Class Classification (OCC) has been used to learn the behavior of single class, which is commonly normal traffic, to detect any deviation from it. However when applying this technique on network as a whole it suffers from the high dimensional network feature spaces. Also, problems may arise when large differences in density exist.

To overcome these problems, we proposed a primary OCC model based on the standard deviation of service's normal behavior. Through this model we dealt with each network service as single class instead of dealing with all network services as a single class. By this way we use just the relevant features of each service, hence reducing the high dimensional network feature spaces and also ensure that each class has - a proximately - uniform distribution.

We summarized our related works in Table 3.1.

Table 3.1 Related Works Summary

| Category | Related Work | Method & Detection technique | Detection Rate |
|---|---|---|---|
| **Supervised** | Chandolikar [65] | Rule induction | 96.18% |
| | Almutairi [66] | Decision tree and Bayesian techniques | 99.95% |
| | Xie [67] | Support vector machine (SVM) | 99.03% |
| | Su [68] | Weighted KNN classifiers | 96.25% |
| | Dartigue [69] | Ensemble of C4.5 binary classifiers | 92.3% |
| | Abd-Eldayem [70] | Naïve Bayes classifier | 99% |
| **Unsupervised** | Amoli [48] | Outlier correlation from DBSCAN clusters | N/A |
| | Jiang [50] | Outlier factor for cluster labeling | 98.5% – 98.6% |
| | Bhuyan [44] | Tree based subspace clustering technique | 89.3% - 99.1% |
| | Hameed [46] | Fuzzy C Means (FCM) | 99% |
| | Leung [47] | Density based and grid based clustering | 97.3% |
| | Han [71] | K-MEANS and information entropy | 95.9% |
| **Semi-Supervised** | Chen [75] | Based on Graph Transducer and Gaussian Fields classifiers and MPCK-means clustering algorithms | N/A |
| | Li [49] | Tri-training SVM | N/A |
| | Wagh [76] | Self-learning algorithm (SLA) | 66.6%-99.25% |
| | Yang [77] | K-means clustering and classification based on distance | 10.44%-99.5% |
| | Li [78] | Semi-supervised fuzzy C-Means clustering | 85.50% |
| | Mahajan [79] | K-means for clustering and probabilistic assignment for labeling generated clusters | 94.7% |
| **OCC** | We [16] | Standard deviation of Transport protocol normal class | 87.7%-99.2% |
| | Araki [57] | Multistage OCSVM | 94% |
| | Winter [58] | OCSVM based on malicious class | 98% |
| | Giacinto [59] | Modular Multiple Classifier System OCSVM | 94% |
| | Ma [73] | Dissimilarity-based one-class classifiers OCSVM | 95%-98% |
| | Zainal [74] | ensemble of Genetic Programming (LGP), Adaptive Neural Fuzzy Inference System (ANFIS) and Random Forest (RF) one-class classifiers | 88%-97.43% |

# Chapter 4: Real Dataset Collection

NIDSs, particularly in anomaly-based approaches, suffer from accurate evaluation, comparison and deployment which originates from the scarcity of adequate datasets. Many of datasets are internal and cannot be shared due to privacy issues, others are heavily anonymized and do not reflect current trends, or they lack certain statistical characteristics. These deficiencies are primarily the reasons why we need to test and evaluate our primary proposed model on a real traffic dataset.

We have chosen Alaqsa University network to collect the traffic coming to and going from its web server, found under http://www.alaqsa.edu.ps. We named this dataset as BM-AUN2015 which refers to the authors of this dataset who are Barhoom and Matar and AUN2015 refers to Alaqsa University Network 2015.

## *4.1 Real Traffic Collection*

The first step in our data collection is to collect the real traffic coming from and going to the web server which is hosting Alaqsa university website. The web server, as shown in Figure 4.1, is located in the DMZ (Demilitarized Zone). The operating system of the web server is Windows Server 2012 R2 64bit and installed on it IIS 8 web server. The Web Server has 4 CPU and 8 GB RAM with NIC speed of 1 Gb/s and connected to a 1Gb/s switch.

A packet sniffer Server has RAM of 4 GB, 2 CPU, 1 Gb/s NIC with promiscuous mode enabled and installed on it WireShark v1.12 software. This server is connected to the mirror port of the same switch at which the web server is connected. The Web Server and WireShark server are operating in the DMZ behind a hardware firewall, called FortiGate.

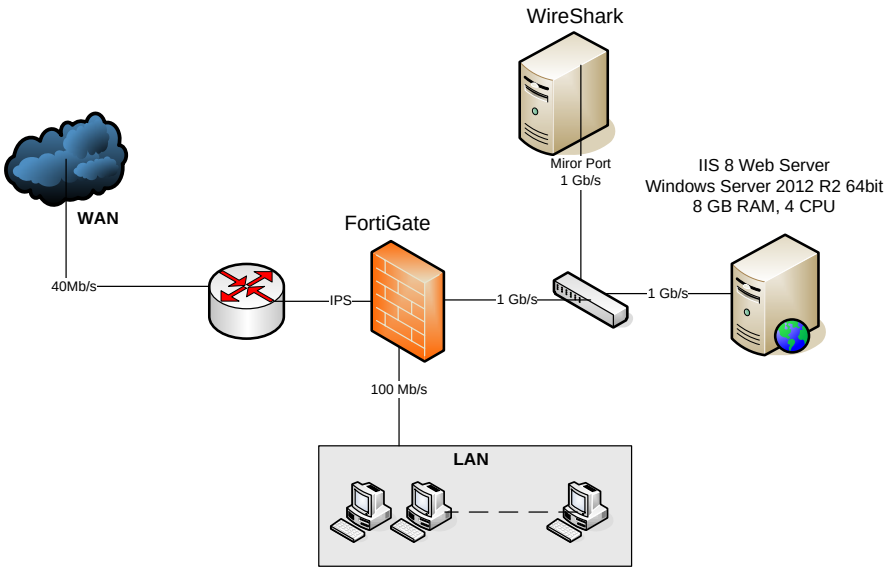


**Figure 4.1 Alaqsa University web server and its network infrastructure**

The FortiGate firewall has three configured terminals, one of them is connected to the WAN through a Cisco router and activated on it an Intrusion Prevention System IPs, the second is connected to the LAN and the third is connected to the DMZ at which the Web server and WireShark Server are located.

Real traffic is not guaranteed to be free from intrusion traces but we have a guarantee in highly percentage that the traffic coming to the Web server is normal because of the IPS, also all of the internal PC's have antivirus program installed on them.

We have captured the traffic that going to and coming from the web server in different days and different time and different period as shown in Table 4.1.

**Table 4.1 Real traffic Capture statistics**

| Date | Start time | End time | Capture time | #Sessions | #Packets |
|---|---|---|---|---|---|
| Day1 | 2:14:55 PM | 2:34:25 PM | 0:19:30 | 703 | 89,406 |
| Day2 | 1:28:31 PM | 2:39:54 PM | 1:11:23 | 3,296 | 475,057 |
| Day3 | 8:33:32 AM | 2:26:28 PM | 5:52:56 | 13,548 | 2,132,748 |
| Day4 | 11:19:42 AM | 2:08:05 PM | 2:48:23 | 10,332 | 13,62,168 |
| | | | | 27,879 | 4,059,379 |

## *4.2 Attack Traffic Collection*

In order to collect the packets of an attack traffic and to be sure that these packets are purely came from an attack source and because that we cannot perform the attack operations on the university production website server, we have prepared a secondary Web Server with the same settings and properties as the production Web Server and the same network topology and infrastructure. As shown in Figure 4.2.

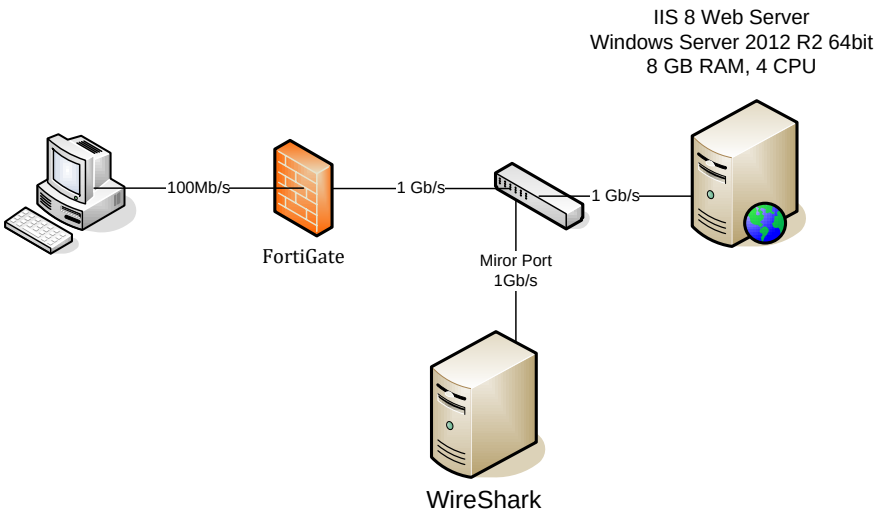


**Figure 4.2 Secondary Web Server to perform attack operations**

### 4.2.1 The Performed Attacks

We have chosen four DoS attack types two of them target the transport protocol and the other two types target the application protocol; in our case we chose attacks that target TCP protocol as the transport protocol for HTTP protocol and the HTTP protocol as the application protocol.

#### *4.2.1.1 Transport Layer DoS Attacks*

We have choosen two DoS attacks that exploit the vulnerabilities of TCP protocol which are:

1. SyncFlood attack
2. SockStress attack

These attack types have been chosen because of many reasons depends on the vulnerabilities of the TCP protocol that it exploits. The reasons are described for each attack as follows:

**Performing SyncFlood Attack**

SyncFlood attack as mentioned in the literature review in chapter 2 exploits the three-way handshake. An incomplete three-way handshaking can occurs normally because of many reasons, for example power shortage in the client side, or network failure occurs and other situations may occurs normally without the intention of doing harm to the server from the client. But what if the client seeks to exploit this vulnerability of TCP protocol to take the server down from a randomly spoofed IP! So we need to test the ability of our primary proposed model of detecting such attacks.

We have chosen hping3 [80] testing tool to perform this type of attack. Hping is a command-line oriented used as packet generator and analyzer for the TCP/IP protocol. It supports TCP, UDP, ICMP and RAW-IP protocols. For more information about this tool, please visit its official website http://www.hping.org, and for more information about how to use this tool visit the following url: http://www.hping.org/manpage.html

We have performed two scenarios of SyncFlood attack and collect each scenario packets separately. After each scenario we have restarted the server to clean up any reserved resources. These attacks operations have been performed on Ubuntu operating system.

**Scenario 1: performing SyncFlood attack from a single IP**

We have performed SyncFlood attack from a single PC using single IP which target the web server. The attack lasts 50 seconds. within this period 1,030,717 packets have been captured and 65,540 Sync messages have been sent from the client as shown in table 4.2.

| **sudo hping3 -c 10000 -d 120 -S -w 64 -p 80 --flood --rand-source 10.10.250.1** |
|---|

Where **sudo** an Ubuntu command to give permissions to **hping3** which is the name of the application binary, **-c 100000** is the number of packets to send, **-d 120** is the size of each

packet that was sent to target machine, **-S** means I am sending SYN packets only, **-w 64** is TCP window size, **-p 80** = Destination port (80 being HTTP port). You can use any port here, **--flood** means Sending packets as fast as possible, without taking care to show incoming replies. Flood mode**, and 10.10.250.1** is Destination IP address.

**Scenario 2: performing SyncFlood attack from a spoofed IP**

We have performed SyncFlood attack from a single PC using spoofed IP which target the web server. The attack lasts 59 seconds. within this period 1,225,823 packets have been captured and 462,206 Sync messages have been sent from the client as shown in table 4.2.

**sudo hping3 -c 10000 -d 120 -S -w 64 -p 21 --flood --rand-source 10.10.250.1**

Where **--rand-source** means using Random Source IP Addresses.

**Table 4.2 Attack types and their capture period**

| Capture time | Attack Type | #Sessions | #Packets | # Instances |
|---|---|---|---|---|
| 0:03:51 | SlowHeader-Senario1 | 2,949 | 371,885 | 2,949 |
| 0:03:56 | SlowHeader- Senario2 | 2,980 | 375,175 | 2,980 |
| 0:04:01 | SlowHeader - Senario3 | 20 | 2,055 | 218 |
| 0:03:98 | SlowHeader - Senario4 | 5 | 543 | 2832 |
| 0:04:03 | SlowPost- Senario1 | 2,977 | 370,561 | 2,977 |
| 0:04:01 | SlowPost- Senario2 | 2,983 | 376,173 | 2,983 |
| 0:03:57 | SlowPost- Senario3 | 20 | 2,539 | 262 |
| 0:04:00 | SlowPost- Senario4 | 5 | 615 | 193 |
| 0:03:44 | SlowRead- Senario1 | 2,960 | 82,593 | 2,960 |
| 0:04:02 | SlowRead- Senario2 | 2,995 | 93,706 | 2,995 |
| 0:04:03 | SlowRead- Senario3 | 20 | 603 | 214 |
| 0:04:07 | SlowRead- Senario4 | 5 | 157 | 116 |
| 0:00:50 | SockStress20thread | 1,319 | 8,560 | 1,929 |
| 0:08:18 | SockStress40thread | 5,584 | 23,640 | 6,876 |
| 0:00:59 | SYNC-FLOOD-SpoofedIP | 462,206 | 1,225,823 | 462,206 |
| 0:01:18 | SYNC-FLOOD | 65,540 | 1,030,717 | 65,540 |

## Performing SockStress Attack

SockStress attack as mentioned in the literature review in chapter 2 exploits the TCP window size . A zero-window size or a very small window size can occurs normally because of network congestion without the intention of doing harm to the server from the client. But what if the client seeks to exploit this vulnerability of TCP protocol to take the server down! This attack is doing complete three-way handshake, which means that it cannot be performed

from a spoofed IP address. So we need to test the ability of our primary proposed model of detecting such attacks.

We have chosen SockStress testing tool [81] to perform this type of attack. This tools is written in Python programming language. We have performed two scenarios of SockStress attack and collect each scenario packets separately. After each scenario we have restarted the server to clean up any reserved resources.

| **Usage - ./sock_stress.py [Target-IP] [Port Number] [Threads]** |
|---|

Where Threads is the number of connections created every one second.

These attacks operations have been performed on Ubuntu operating system.

**Scenario 1: performing DoS-SockStress attack using 20 concurrent connections**

We have performed SockStress attack from a single client. The attack last one minute and 59 seconds. within this period 8,560 packets have been captured and 1,319 TCP sessions have been opened from the five clients as shown in table 4.2.

| **sudo ./sock_stress.py 10.10.250.1 80 20** |
|---|

**Scenario 2: performing DoS-SockStress attack using 40 concurrent connections**

We have performed SockStress attack from a single client. The attack last eight minutes 18 seconds. within this period 23,640 packets have been captured and 5,584 TCP sessions have been opened from the five clients as shown in table 4.2.

| **sudo ./sock_stress.py 10.10.250.1 80 40** |
|---|

#### *4.2.1.2 Application Layer DoS Attacks*

We have choosen three DoS attacks that exploit the vulnerabilities of HTTP protocol as described before in chapter 2 which are:

1. SlowHeader attack
2. SlowPost attack
3. SlowRead attack

These attack types raised recently and have been chosen because of many reasons depends on the vulnerabilities of the HTTP protocol that it exploits, the reasons are described for each attack. We have chosen testing tool called SlowHttpTest [82]. For more information about this tool please visit the following web page https://code.google.com/p/slowhttptest/ , there you can find also test results of popular HTTP servers using this penetration tool. We have installed this tool on Ubuntu operating system. The tool installation and usage can be found under https://code.google.com/p/slowhttptest/wiki/InstallationAndUsage.

### Performing SlowHeader Attack

SlowHeader attack as mentioned in the literature review in chapter 2 exploits the HTTP header behavior. An incomplete HTTP header can occurs normally, for example when there exists large http header that cannot be sent in single packet because of network congestion, or other situations may occurs normally without the intention of doing harm to the server from the client. But what if the client seeks to exploit this vulnerability of HTTP protocol to take the server down! This attack is doing complete three-way handshake, which means that it cannot be performed from a spoofed IP address. So we need to test the ability of our primary proposed model of detecting such attacks. We have performed two scenarios of SlowHeader attack and collect each scenario packets separately. After each scenario we have restarted the server to clean up any reserved resources.

**Scenario 1: performing SlowHeader attack with connection rate 200/sec and max length of follow up data of HTTP header payload is 4 bytes.**

We have performed SlowHeader attack from a single PC which target an Apache web server 2.2.8, as mentioned before in chapter 2, IIS web server is not vulnerable to this attack. The attack last four minutes. within this period 371,885 packets have been captured and 2,949 sessions have been opened from the client as shown in table 4.2.

```
Sudo ./slowhttptest -c 3000 -H my_header_stats -l 240 -i 2 -r 200 -w 512 -y
1024 -t GET -x 4 -p 3 -u http://10.10.250.1/default.aspx
```

Where **-c 3000** is the max number of opened connections, **-H my_header_stats** is to perform SlowHeader attack, **-l 240** is the test duration in seconds, **-i 2** is interval between follow up data in seconds, per connection, **-r 200** is the number of creating connections every one second Initial SYN packet for every connection would have random advertised window size value between **-w 512** and **-y 1024**. **-t GET** is the HTTP method type, **-x** 4 is the max length in bytes of follow up data and **-p 3** is timeout to wait for HTTP response on probe connection, after which server is considered inaccessible.

**Scenario 2: performing SlowHeader attack with connection rate 50/sec and max length of follow up data of HTTP header payload is 8 bytes.**

We have performed SlowHeader attack from a single PC which target an Apache web server 2.2.8. The attack last four minutes. within this period 375,175 packets have been captured and 2980 sessions have been opened from the client as shown in table 4.2.

```
Sudo ./slowhttptest -c 3000 -H my_header_stats -l 240 -i 2 -r 50 -w 512 -y
1024 -t GET -x 8 -p 3 -u http://10.10.250.1/default.aspx
```

**Scenario 3: performing SlowHeader attack with connection rate 1/sec, max length of follow up data of HTTP header payload is 4 bytes and maximum of 20 connections.**

We have performed SlowHeader attack from a single PC which target an Apache web server 2.2.8. The attack last four minutes, through this period a new connection opened each second with 20 maximum opened connections. Within this period 2,055 packets have been captured and 20 sessions have been opened from the client as shown in table 4.2.

```
Sudo ./slowhttptest -c 20 -H my_header_stats -l 240 -i 2 -r 1 -w 512 -y 1024
-t GET -x 4 -p 3 -u http://10.10.250.1/default.aspx
```

**Scenario 4: performing SlowHeader attack with connection rate 1/sec, max length of follow up data of HTTP header payload is 4 bytes and maximum of 5 connections.**

We have performed SlowHeader attack from a five PC's which target an Apache web server 2.2.8. The attack last four minutes, through this period a new connection opened each second with 5 maximum opened connections.. within this period 543 packets have been captured and 5 sessions have been opened from the client as shown in table 4.2.

```
Sudo ./slowhttptest -c 5 -H my_header_stats -l 240 -i 2 -r 1 -w 512 -y 1024
-t GET -x 4 -p 3 -u http://10.10.250.1/default.aspx
```

### Performing SlowPost Attack

SlowPost attack as mentioned in the literature review in chapter 2 exploits the HTTP body behavior. Time required to complete HTTP body varies depends on the body size, e.g. attach a large file, also it depends on the network congestion which is specified by window size in the TCP protocol. These situations can occur normally without any intention of doing harm to the server from the client. But what if the client seeks to exploit this vulnerability of HTTP protocol to take the server down by reserve its resources for long time! This attack is doing complete three-way handshake, which means that it cannot be performed from a spoofed IP address. So we need to test the ability of our primary proposed model of detecting such attacks. We have performed two scenarios of SlowPost attack and collect each scenario packets separately. After each scenario we have restarted the server to clean up any reserved resources.

**Scenario 1: performing SlowPost attack with connection rate 200/sec and length of follow up data of HTTP body payload is 1 byte.**

We have performed SlowPost attack using single pc which target the web server. The attack is configured to creates 200 connections each second. Each established http session waits 2 seconds between each packet before sending the next payload. The attack last four minutes. within this period 370,561 packets have been captured and 2,977 sessions have been opened from the client as shown in table 4.2.

```
Sudo ./slowhttptest -c 3000 -B my_body_stats -l 240 -i 2 -r 200 -w 512 -y 1024
-s 8192 -t POST -x 1 -p 3 -u http://10.10.250.1/default.aspx
```

Where **-c 3000** is the max number of opened connections, **-B my_body_stats** is to perform SlowPost attack, **-l 300** is the test duration in seconds, **-i 2** is interval between follow up data in seconds, per connection, **-r 200** is the number of creating connections every one second. Initial SYN packet for every connection would have random advertised window size value between **-w 512** and **-y 1024**. **-s 8192** is the value of Content-Length header, **-t POST** is the

HTTP method type, **-x 1** is the max length of follow up data which is one byte and **-p 3** is timeout to wait for HTTP response on probe connection, after which server is considered inaccessible.

**Scenario 2: performing SlowPost attack with connection rate 50/sec and max length of follow up data of HTTP body payload is 32 bytes.**

We have performed SlowPost attack using single pc which target the web server. The attack is configured to creates 50 connections each second. Each established http session waits 2 seconds between each packet before sending the next payload. The attack last four minutes. within this period 376,173 packets have been captured and 2,983 sessions have been opened from the client as shown in table 4.2.

```
Sudo ./slowhttptest -c 3000 -B my_body_stats -l 240 -i 2 -r 50 -w 512 -y 1024
-s 8192 -t POST -x 32 -p 3 -u http://10.10.250.1/default.aspx
```

**Scenario 3: performing SlowPost attack with connection rate 1/sec, max length of follow up data of HTTP body payload is 1 bytes and maximum of 20 connections.**
We have performed SlowPost attack using single pc which target the web server. The attack is configured to creates one connection each second with 20 maximum opened connections.. Each established http session waits 2 seconds between each packet before sending the next payload. The attack last four minutes. within this period 2,539 packets have been captured and 20 sessions have been opened from the client as shown in table 4.2.

```
Sudo ./slowhttptest -c 20 -B my_body_stats -l 240 -i 2 -r 1 -w 512 -y 1024  -s
8192 -t POST -x 1 -p 3 -u http://10.10.250.1/default.aspx
```

**Scenario 4: performing SlowPost attack with connection rate 1/sec, max length of follow up data of HTTP body payload is 1 bytes and maximum of 5 connections.**
We have performed SlowPost attack using single pc which target the web server. The attack is configured to creates one connection each second with 5 maximum opened connections. Each established http session waits 2 seconds between each packet before sending the next payload. The attack last four minutes. within this period 615 packets have been captured and 5 sessions have been opened from the client as shown in table 4.2.

```
Sudo ./slowhttptest -c 5 -B my_body_stats -l 240 -i 2 -r 1 -w 512 -y 1024  -s
8192 -t POST -x 1 -p 3 -u http://10.10.250.1/default.aspx
```

### Performing SlowRead Attack

SlowRead attack as mentioned in the literature review in chapter 2 exploits the HTTP request behavior. Time required to complete HTTP file retrieve varies depends on the file size, e.g. a large file, also it depends on the network congestion which is specified by window size in the TCP protocol. These situations can occur normally without any intention of doing harm to the server from the client. But what if the client seeks to exploit this vulnerability of HTTP

protocol to take the server down by reserve its resources for long time! This attack is doing complete three-way handshake, which means that it cannot be performed from a spoofed IP address. So we need to test the ability of our primary proposed model of detecting such attacks. We have performed two scenarios of SlowPost attack and collect each scenario packets separately. After each scenario we have restarted the server to clean up any reserved resources.

**Scenario 1: performing SlowRead attack with connection rate 200/sec and 32 bytes to read from receive buffer with single read() operation.**

We have performed SlowRead attack using single pc which target the web server. The attack is configured to creates a total of 3000 connections, with 200 connections every one second. The attack last four minutes. within this period 82,593 packets have been captured and 2,960 TCP sessions have been opened as shown in table 4.2.

```
Sudo ./slowhttptest -c 3000 -X slow_read_stats -l 240 -r 200 -w 512 -y 1024 -
n 5 -z 32 -p 3 -u http://10.10.250.1/default.aspx
```

Where **slow_read_stats** is to enable slow read mode, **-X** starts Slow Read test with **–c 3000** connections, creating **–r 200** connections per second. Initial SYN packet for every connection would have random advertised window size value between **-w 512** and **-y 1024**, and application would read **-z 32** bytes every **-n 5** seconds from each socket's receive buffer. To multiply overall response size. Probe connection would consider server DoSed, if no response was received after **-p 3** seconds.

**Scenario 2: performing SlowRead attack with connection rate 50/sec and 64 bytes to read from receive buffer with single read() operation.**

We have performed SlowRead attack using single pc which target the web server. The attack is configured to creates a total of 3000 connections, with 50 connections every one second. The attack last four minutes. within this period 93.7046 packets have been captured and 2,995 TCP sessions have been opened as shown in table 4.2.

```
Sudo ./slowhttptest -c 3000 -X slow_read_stats -l 240 -r 50 -w 512 -y 1024 -n
5 -z 64 -p 3 -u http://10.10.250.1/default.aspx
```

**Scenario 3: performing SlowRead attack with connection rate 1/sec, 32 bytes to read from receive buffer with single read() operation and maximum of 20 connections.**

We have performed SlowRead attack using single pc which target the web server. The attack is configured to creates a total of 20 connections, with 1 connection every one second. The attack last four minutes. within this period 603 packets have been captured and 20 TCP sessions have been opened as shown in table 4.2.

```
Sudo ./slowhttptest -c 20 -X slow_read_stats -l 240 -r 1 -w 512 -y 1024 -n 5
-z 32 -p 3 -u http://10.10.250.1/default.aspx
```

**Scenario 4: performing SlowRead attack with connection rate 1/sec, 32 bytes to read from receive buffer with single read() operation and maximum of 5 connections.**
We have performed SlowRead attack using single pc which target the web server. The attack is configured to creates a total of 5 connections, with 1 connection every one second. The attack last four minutes. within this period 157 packets have been captured and 5 TCP sessions have been opened as shown in table 4.2.

```
Sudo ./slowhttptest -c 5 -X slow_read_stats -l 240 -r 1 -w 512 -y 1024 -n 5 -
z 32 -p 3 -u http://10.10.250.1/default.aspx
```

We have performed the previous mentioned attacks in different period depends on packets generated/second. For example SYNC-FLOOD-SpoofedIP is performed for just 59 second which generate 1,225,823 packets and 462,206 sessions within just 59 second as listed in Table 4.2.

## *4.3 Importing PCAP Files into Oracle Database*

The captured packets using Wireshark program are saved as PCAP files, we need to extract packets from these files and import them to Oracle database in order to generate the required features. To do so we have coded JAVA classes that use jNetPcap v1.4 [83] package which aim to extract the packets from PCAP files. Also we have used a package to parse HTTP protocol in order to extract HTTP entities, this package is called ApacheHttpcomponents v4.4.1 [84]. We have extracted the TCP and HTTP attributes and also other generated attributes and stored them in Oracle database table. These attributes are listed in Table 4.2.

**Table 4.3 database table of fileds extracted from the Packets in the PCAP file**

| # | Column name | Data type | Description |
|---|---|---|---|
| 1 | FLOWKEY | Number | TCP session ID |
| 2 | PACKET_NUMBER | Number | Packet Number |
| 3 | PACKET_TIME_STAMP | Timestamp | Packet time stamp |
| 4 | SOURCE_IP | String | Packet source IP |
| 5 | SRC_PORT | Number | Packet source port |
| 6 | DESTINATION_IP | String | Packet destination IP |
| 7 | DST_PORT | Number | Packet destination port |
| 8 | PACKET_SIZE | Number | Packet size |
| 9 | IS_CHECKSUM_VALID | Boolean | TCP packet checksum valid? |
| 10 | TCP_FLAGS_PSH | Boolean | Does TCP PSH flag is set |
| 11 | TCP_FLAGS_ACK | Boolean | Does TCP ACK flag is set |
| 12 | TCP_FLAGS_RST | Boolean | Does TCP RST flag is set |
| 13 | TCP_FLAGS_SYN | Boolean | Does TCP SYN flag is set |
| 14 | TCP_FLAGS_FIN | Boolean | Does TCP FIN flag is set |
| 15 | TCP_WINDOW_SIZE | Number | TCP window size |
| 16 | IS_HTTP_SESSION | Boolean | Does client initiate HTTP session |
| 17 | HTTP_HEADER_LENGTH | Number | The length of the HTP header |

| | | | |
|---|---|---|---|
| 18 | HTTP_PAYLOAD_LENGTH | Number | The length of the HTTP body |
| 19 | HTTP_REQ_REFERER | String | HTTP Request referrer |
| 20 | HTTP_USER_AGENT | String | The client user-agent |
| 23 | HTTP_RESPONSE_CODE | Number | The response code received from the server |
| 24 | TCP_PAYLOAD_LENGTH | Number | The TCP payload length |
| 25 | PACKET_TIME_NANO_SEC | Number | The time in Nano second |
| 26 | AVG_TIME_HTTP_HEADER_COMPLETE | Real | The avg time to complete HTTP header in current TCP session |
| 27 | NUMBER_OF_HTTP_HEADERS | Number | Number of HTTP headers sent from client in current TCP session |
| 28 | IS_HTTP_HEADER_END | Boolean | Does client HTTP header sent completely |
| 29 | SERVER_TIME_WINDOW_2SEC | String | A unique Time window label for every 2Sec |
| 30 | SERVER_TIME_WINDOW_4SEC | String | A unique Time window label for every 4Sec |
| 31 | Label | String | Type of packet, Normal/Attack type |

## *4.4 Collected Traffic Statistics*

In this section we have generate some charts which show the connection behavior and number of connections established with the server per second, we also show another charts that presents the cumulative number of open connections with the server. We have also figure out the normal user's connection behavior. The reason of traffic statistics' charts is to understand the normal and attack behavior.

### 4.4.1 Normal Traffic Connection's Behavior

We have collected traffic coming from and going to production web server of Alaqsa university in different four days as shown in Table 4.1. The following figures show the connections per second established with web server and also the cumulative of active connections opened through time in seconds in web server. Figure 4.3, Figure 4.4, Figure 4.5 and Figure 4.6 show the connections established with the production web server through time in seconds in the days Day1,Day2, Day3 and Day4 respectively. Figure 4.7 shows the normal connection's behavior to the web server from a single user.

As shown in Figure 4.3, which presents the connections to the server through time in seconds, the number of connections established concurrently per second ranging from 3 to 10 connections in average as shown in Figure 4.3(A). Note the gapes in the second timeline in Figure 4.3(A) which means that there's zero connections at this moment. The cumulative of active connections through time is presented in Figure 4.3(B) is ranging from 0 to 10 connections in average which means that at any given time we can find just 1 to 20 connection opened. This capture, as shown in Table 4.1, was taken in 20 minutes period of time at 14:15 which is the last hour of the working day.

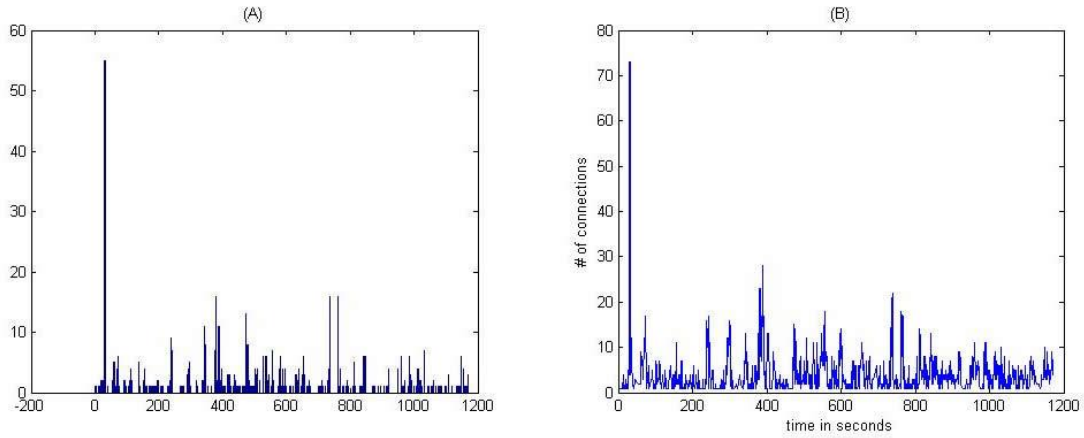


**Figure 4.3 Histogram of Normal connections in Day1**

(A) Connections created per second, (B) Cumulative of active connections in seconds

Figure 4.4 shows the connections to the server captured in a period of one hour began at 13:20 in the second day. From the figure we can see that the number of connections established concurrently per second ranging from 0 to 10 connections in average as shown in Figure 4.4(A). The cumulative of the opened connections through time is presented in Figure 4.4(B) ranging from 1 to 20 connections in average which means that at any given time we can find just 5 to 20 connection opened.

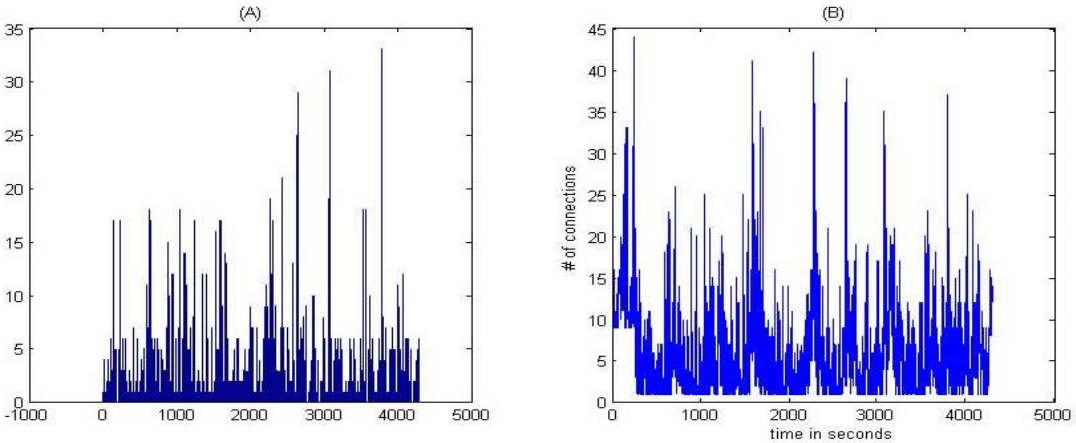


**Figure 4.4 Histogram of Normal connections in Day2**

(A) Connections created per second, (B) Cumulative of active connections in seconds

Figure 4.5 also presents the connections established in the third working day, which have been captured in a period of six hours began at 08:30. From the figure we can see that the number of connections established concurrently per second ranging from 0 to 15 connections in average as shown in Figure 4.5(A). The cumulative of the opened connections through time is presented in Figure 4.5(B) ranging also from 0 to 10 connections in average which means that at any given time we can find just 1 to 20 connection opened.

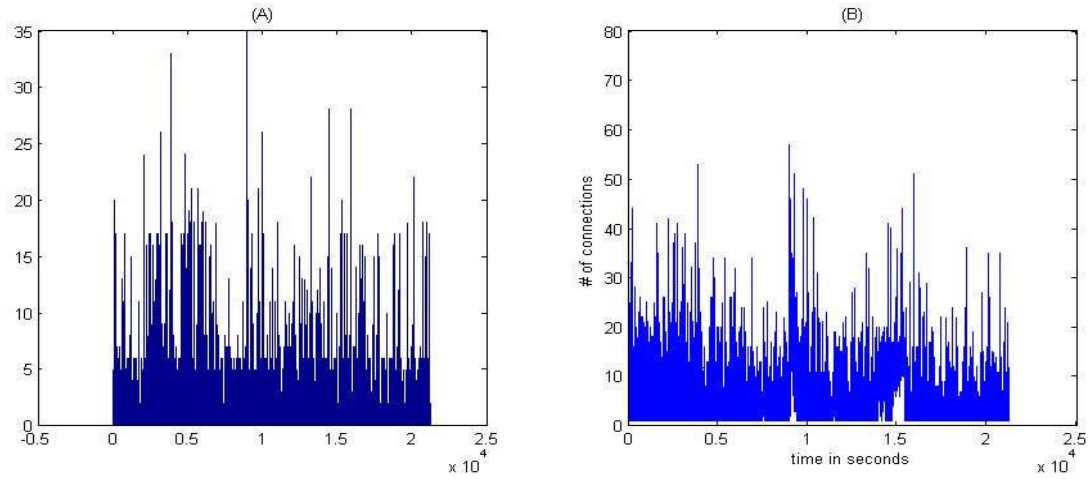


**Figure 4.5 Histogram of Normal connections in Day3**

(A) Connections created per second, (B) Cumulative of active connections in seconds

The connections established in the fourth day which have been capture at 11:19 for a period of three hours is shown in Figure 4.6. The number of connections established concurrently per second ranging from 0 to 10 connections in average as shown in Figure 4.6 (A). The cumulative of the opened connections through time is presented in Figure 4.6 (B) ranging also from 0 to 10 connections in average which means that at any given time we can find just 1 to 25 connection opened.

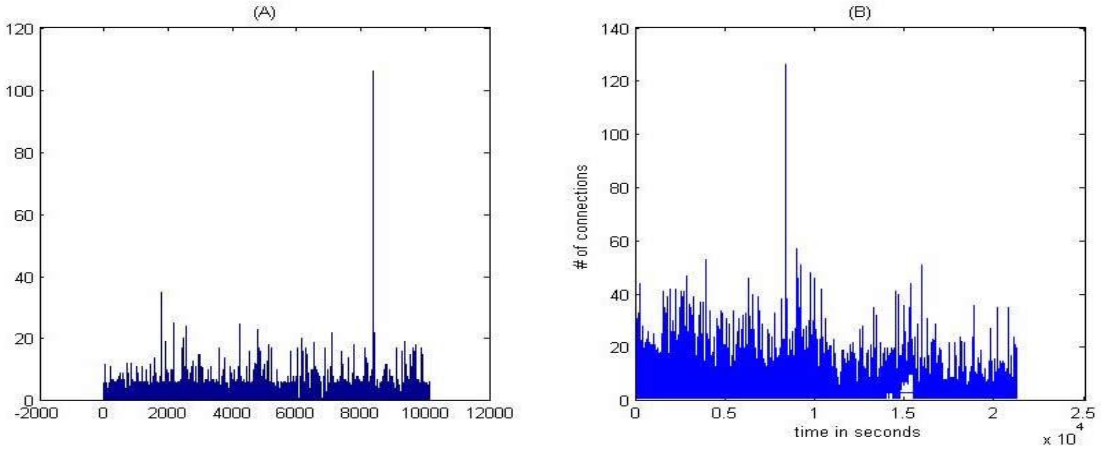


**Figure 4.6 Histogram of Normal connections in Day4**

(A) Connections created per second, (B) Cumulative of active connections in seconds

The normal behavior of a single user connect to the web server is shown in Figure 4.7. As shown in the Figure 4.7 (A) , the user connect to the web server at different periods of time, and the concurrent connections per second ranging from 1 to 6, also the average TCP session time doesn't exceeds 20 seconds.

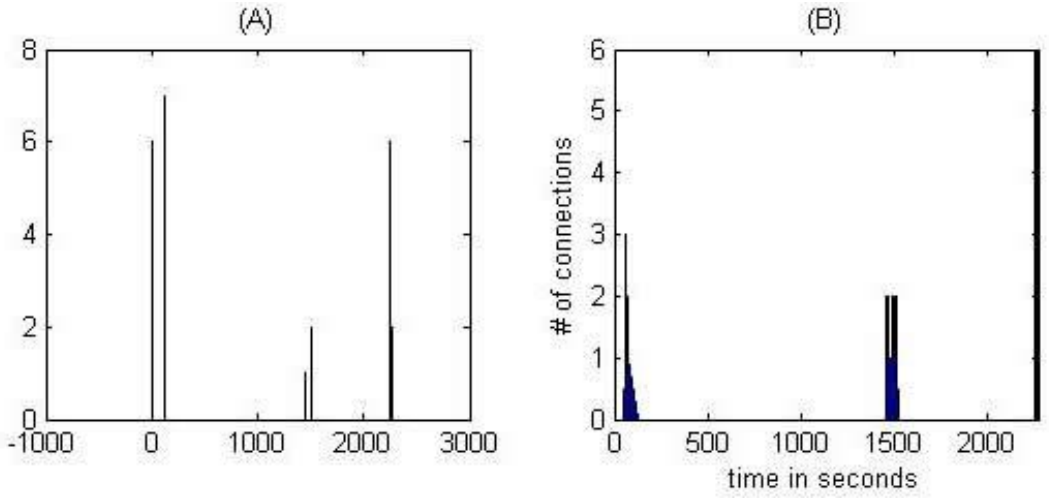


**Figure 4.7 Histogram of Normal connections of single user**

(A) Connections created per second, (B) Cumulative active connections in seconds

## 4.4.2 Attack Traffic Connection's Behavior

In this section we figure out the connection behavior of the attacks performed at the web server as described in section 4.2.

### *4.4.2.1 Network Layer Attacks' Behavior*

SYNC-FLOOD DoS attack connection's behavior as shown in Figure 4.8 has significant difference compared with the connection's behavior of the normal traffic.

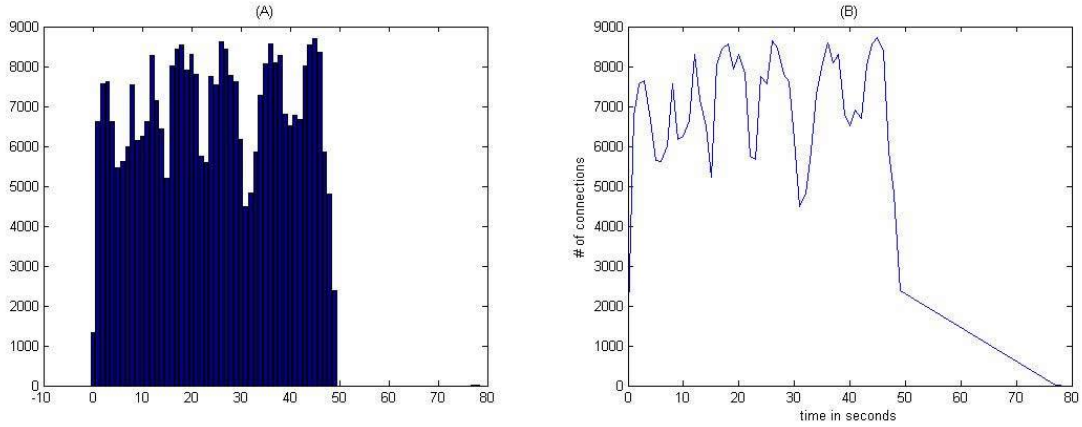


**Figure 4.8 Histogram of SYNC-FLOOD connections**

(A) Connections created per second, (B) Cumulative of active connections in seconds

The number of connections established by the client per second is very high ranging from 5000 to 8000 connection per second as shown in Figure 4.8 (A) and the accumulative opened sessions in average exceeds 6000 connections at any given time as shown in Figure 4.8 (B). The drop of the number of open connections at the last few seconds happened because of ending the attack which closes the opened connections.

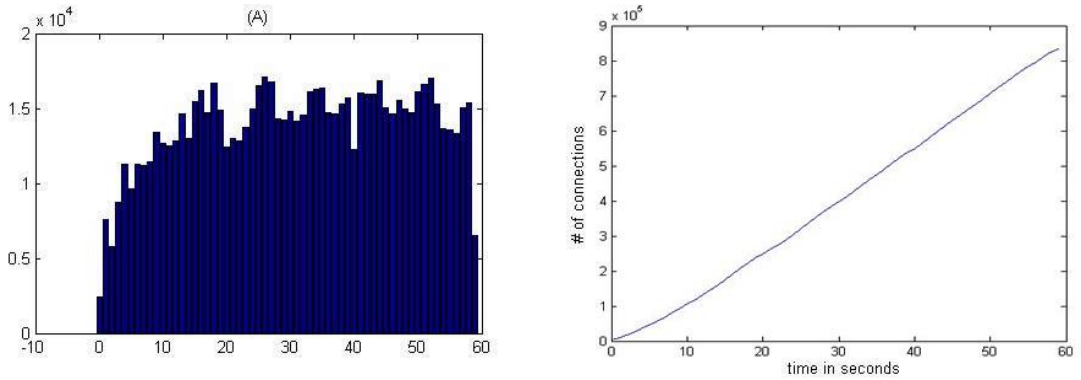


**Figure 4.9 Histogram of SYNC-FLOOD Spoofed IP connections**

(A) Connections created per second, (B) Cumulative of active connections in seconds

SYNC-FLOOD using spoofed IP attack has also a significant difference compared with normal connection's behavior. As shown in Figure 4.9, the number of connections per second to the server is extremely high, which range from 13,000 to 15,000 connections per second as shown in Figure 4.9 (A). On the other hand, the cumulative opened sessions on the server reaches 17,000 open connections per second as shown in Figure 4.9 (B). The drop of the number of open connections at the last few seconds happened because of ending the attack which closes the opened connections.

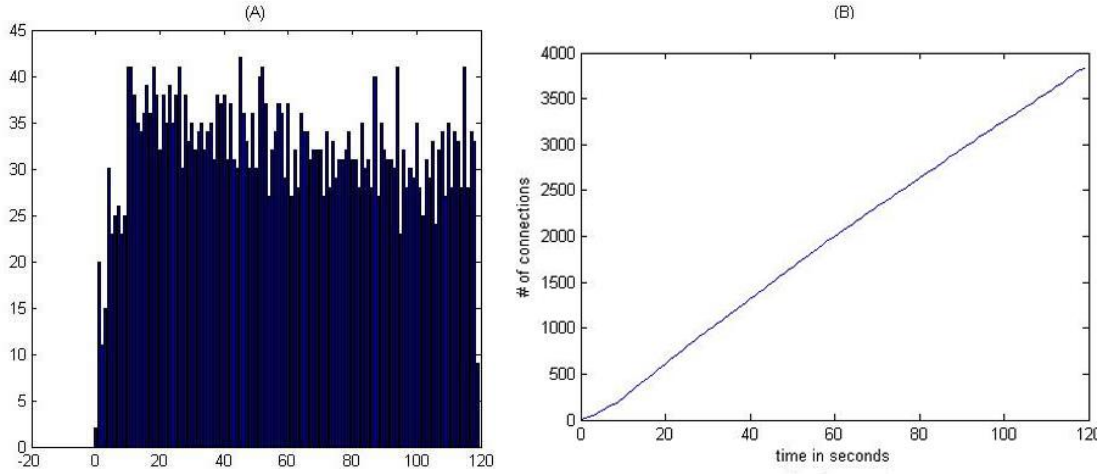


**Figure 4.10 Histogram of SockStress20thread connections**

(A) Connections created per second, (B) Cumulative of active connections in seconds

SockStress attack differs in behavior than Sync-Flood attack that SockStress attack does a complete three-way handshaking as described in chapter 2. As shown in Figure 4.10 (A), the number of established connections per second ranges from 20 to 30 connections per second and the cumulative number of opened sessions ranges from 40 to 50 sessions at any given time as shown in Figure 4.10 (B). This attack opened a 20 threads each of them creates it own connections and handle its connections. Note that based on the connection behavior of normal

traffic, SockStress using 20 threads behaves as the normal connection traffic except that it don't establish HTTP session.

A second SockStress attack scenario, this attack used 40 threads to perform the attack. As shown in Figure 4.11 (A), the number of established connections per second ranges from 30 to 40 connections per second and the cumulative number of opened sessions ranges from 60 to 80 sessions at any given time as shown in Figure 4.11 (B).

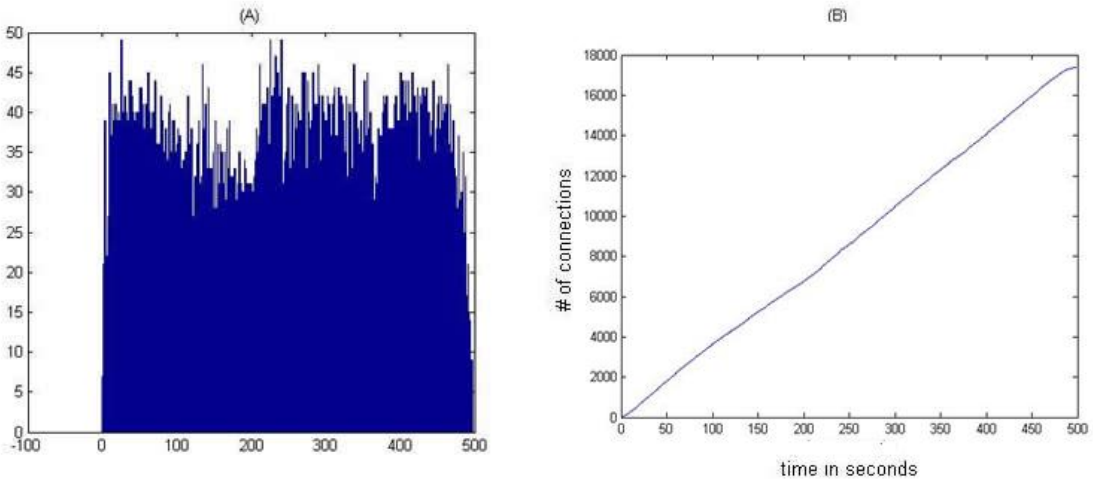


**Figure 4.11 Histogram of SockStress40thread connections**

(A) Connections created per second, (B) Cumulative of active connections in seconds

#### *4.4.2.1 Application Layer Attacks' Behavior*

Application layer attacks' behavior differs than network layer attacks' behavior that it performs legitimate operations but it intends to keep the session open as long as possible as described in chapter 2.

Figure 4.12 shows the connection behavior of a scenario of SlowPost attack. This scenario is described in subsection 4.2.1.2. As shown in Figure 4.12 (A), it seems that it sends a burst of approximately 200 connections every 45 seconds. Figure 4.12 (B) shows the cumulative opened sessions per time of seconds; as shown, the number of opened sessions exceeds every 45 seconds and reaches 1800 concurrent opened sessions.

Figure 4.13 presents the connection activity of a single SlowPost2Sec attack connection, as shown the connection sends in average one packet every 2 seconds to the server in order to keep the connection open as long as possible.

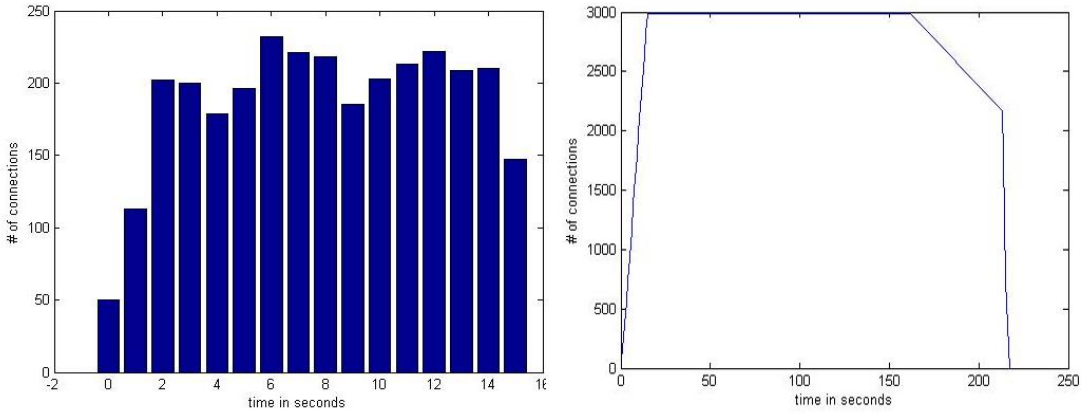


**Figure 4.12 Histogram of SlowPost connections**

(A) Connections created per second, (B) Cumulative of active connections in seconds

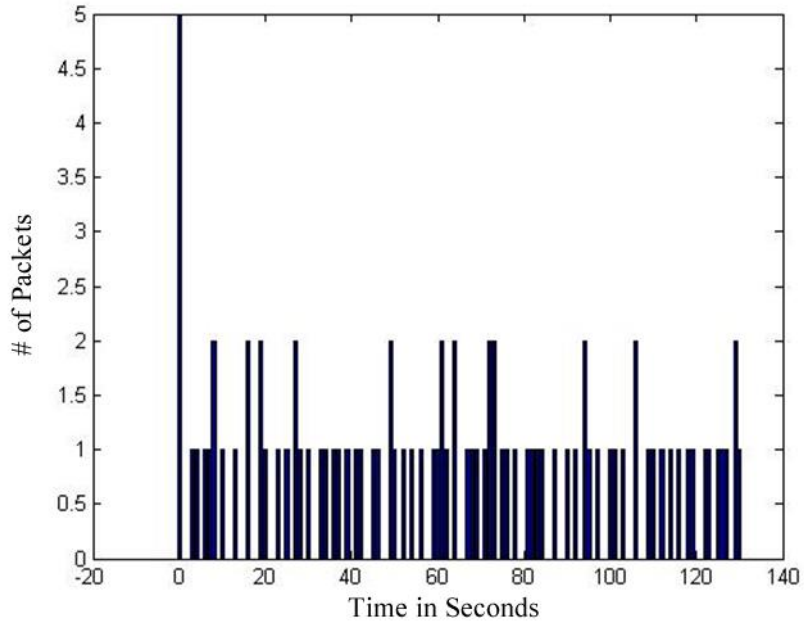


**Figure 4.13 Histogram of single SlowPost connection activity**

## *4.5 Features Generation*

In order to identify the behavior of a client if it behaves normal or abnormal , we need to generate instances with statistics features from both TCP and HTTP protocols in addition to some basic features, these features are generated from Table 4.3. A list of generated features is listed below, these features have been generated using Oracle PL/SQL . These features have been chosen based on the behavior of performed attacks described in chapter 2.

**Table 4.4 Real dataset generated features**

| # | Feature Name | Data type | Description |
|---|---|---|---|
| 1 | FLOWKEY | Integer | The TCP Session ID |
| 2 | CLIENT_REPLY_ACK | Boolean | Does Client complete the handshaking |
| 3 | CLIENT_TIME_TO_REPLY_ACK | Integer | The time to replay at Server SYNC Ack |
| 4 | NUMBER_OF_SERVER_ACK | Integer | Number of Server Ack to Sync message |
| 5 | TCP_SESSION_TIME | Integer | The total time of the TCP session |
| 6 | HTTP_SESSION_TIME | Integer | The total time of the HTTP session |
| 7 | IS_HTTP_SESSION | Boolean | Does this session include an Http session? |
| 8 | IS_HTTP_HEADER_END | Boolean | Does Http Request headers ended? |
| 9 | AVG_TIME_HTTP_HEADER_COMPLETE | Real | Average time to complete Request HTTP header. |
| 10 | NUMBER_OF_CLIENT_HTTP_HEADERS | Integer | # of headers in the same TCP session |
| 11 | IS_CLIENT_FIN_TCP_CONNECTION | Bolean | Does the client end the TCP session |
| 12 | NUMBER_OF_CLIENT_TCP_PSH | Integer | # of client packets with PSHflag set. |
| 13 | AVG_TCP_PAYLOAD_LENGTH | Real | The average length of TCP packet payload. |
| 14 | AVG_CLNT_TCP_WINDOW_SIZE | Real | The average size of the client TCP window |
| 15 | CURRENT_CONNECTIONS_2SEC | Integer | # of connections 2 sec time window |
| 16 | CURRENT_CONNECTIONS_4SEC | Integer | # of connections 4 sec time window |
| 17 | USER_AGENTS_2SEC | Integer | # of distinct user agents used in 2 sec |
| 18 | NUMBER_OF_CIENT_FLOW_SYNC | Integer | # of SYNC sent in current TCP session |
| 19 | NUMBER_ZERO_WINDOW_PKTS | Integer | # of client zero window size packets |
| 20 | CLASS_TYPE | String | Instance class |

Before we extract the most relevant features we need to look at the difference between normal traffic and attack traffic based on these generated features to understand their behavior. We categorize the comparison between these features based on the protocol used. The total number of instances generated are listed in Table 4.5.

As shown in Table 4.5, the total number of normal instances is 27,879 instances, while the total number of attack is 555,611 instances. We have divide these instances into two datasets, one for training and the other for testing purposes as shown in Table 4.6. the remaining normal instances and attack instances shown in Table 4.1 and Table 4.2 which summarized in Table 4.5 are used for model testing purposes.

**Table 4.5 BM-AUN2015 dataset's instances**

| | Normal | | | | Attack | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Application layer attack | | | Network layer attack | |
| | Day1 | Day2 | Day3 | Day4 | Slow Read | Slow Post | Slow Header | Sock Stress | SYN-Flood |
| # instances | 703 | 3296 | 13548 | 10332 | 6360 | 6415 | 6285 | 8805 | 527746 |
| ∑ | 27,879 | | | | 555,611 | | | | |

**Table 4.6 BM-AUN2015 training dataset**

| | Normal | Attack | | |
|---|---|---|---|---|
| | Day3 | SlowPost-Scenario1 | SlowHeader-Scenario1 | SlowRead-Scenario1 |
| # instances | 13,548 | 2,977 | 2,949 | 2,960 |
| ∑ | 13,548 | 8,886 | | |

As shown in Table 4.6, we chose Day3 for the model training because it contains workday activates, as shown in Table 4.1, this day is a collection of 6 hours working day. We also include just the first scenario of application layer attacks. Note that our model is a OCC that need just the normal instances to be learned on, but the attack instances is needed here just to help us to select the most relevant features.

### 4.5.1 Network Layer Attack Features

In this subsection we look deep at the differences between normal traffic and network layer attack traffic, e.g. Sync-flood attack.

**CLIENT_REPLY_ACK**

This feature indicate whether the client send an acknowledgment to the server to complete the three-way handshaking. As described in chapter 2 SyncFlood attacks exploit this vulnerability to keep server waiting for this acknowledgment to establish connection.

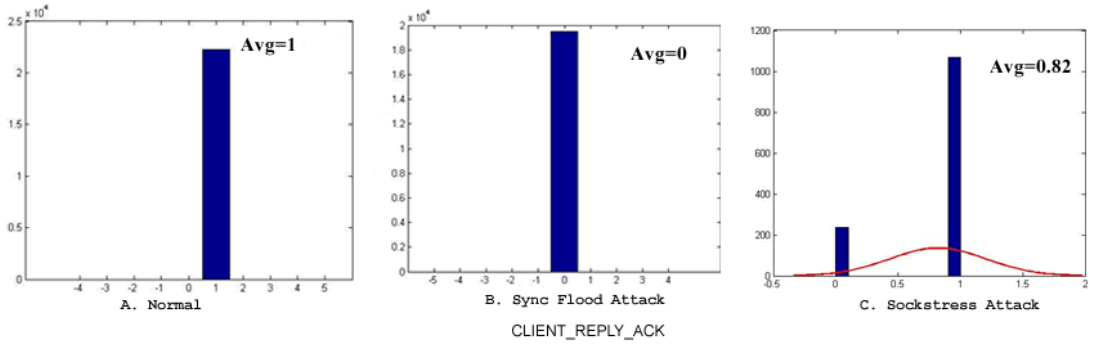


**Figure 4.14 Distribution of CLIENT_REPLY_ACK feature in TCP attacks**

As shown in Figure 4.14, the average of client acknowledgment reply in SyncFlood attack traffic is zero but it's one in normal traffic and also in Sockstress attack traffic.

**CLIENT_TIME_TO_REPLY_ACK**

This feature give us information about the amount of time elapsed before the client complete the three-way handshaking.

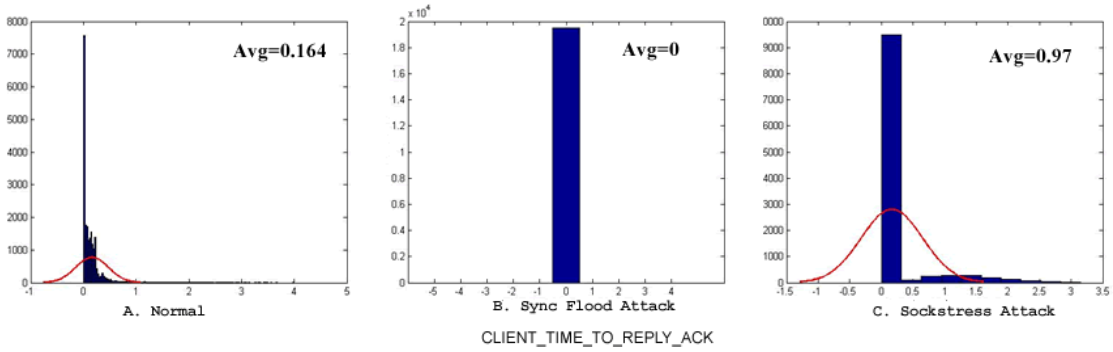


**Figure 4.15 Distribution of CLIENT_TIME_TO_REPLY_ACK feature in TCP attacks**

As shown in Figure 4.15 the average time elapsed to reply acknowledgment by the client in normal traffic is close to zero where it's close to 1 second in SockStress attack which means that SockStress attack waste the server time for 1 second before sending its acknowledgment to complete the three-way handshaking. In SyncFlood attack, the average time elapsed to send acknowledgment is zero which means that SyncFlood didn't send its acknowledgment.

**NUMBER_OF_SERVER_ACK**

This features counts the number of times the server sends sync acknowledgment to the client, which indicates that the client didn't send the last three-way handshaking to establish connection. As shown in Figure 4.16, the number of server Sync acknowledgment in normal traffic is close to zero while in SyncFlood attack it is significantly large, which reaches five acknowledgment for each connection in average, it is also large in SockStress which reaches approximately 3 times in average. As described in chapter 2, SockStress attack complete the three-way handshaking but the large number of server Sync acknowledgment is due to the elapsed time before the client send the last acknowledgment to establish the connection as shown in Figure 4.15.

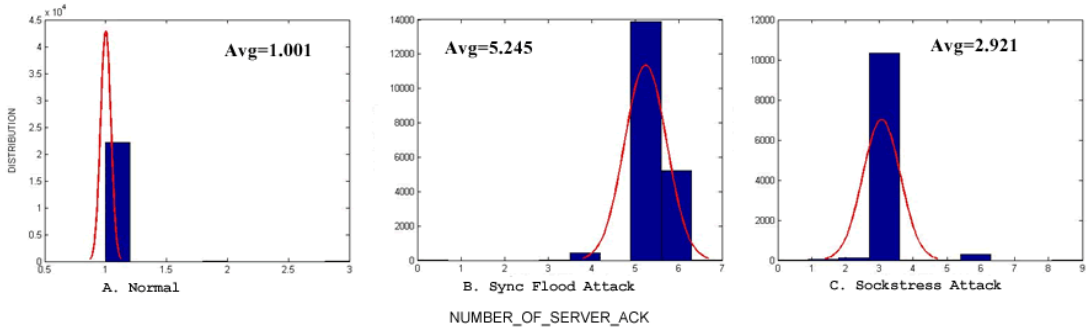


**Figure4.16 Distribution of NUMBER_OF_SERVER_ACK feature in TCP attacks**

**TCP_SESSION_TIME**

TCP session time is the time of the connection before it is closed. As shown in Figure 4.17 the average TCP session time of a single connection of the normal traffic is 86.6 seconds, while it is smaller in SyncFlood and SockStress attacks because it depends on the connection timeout limit in configured on server side.

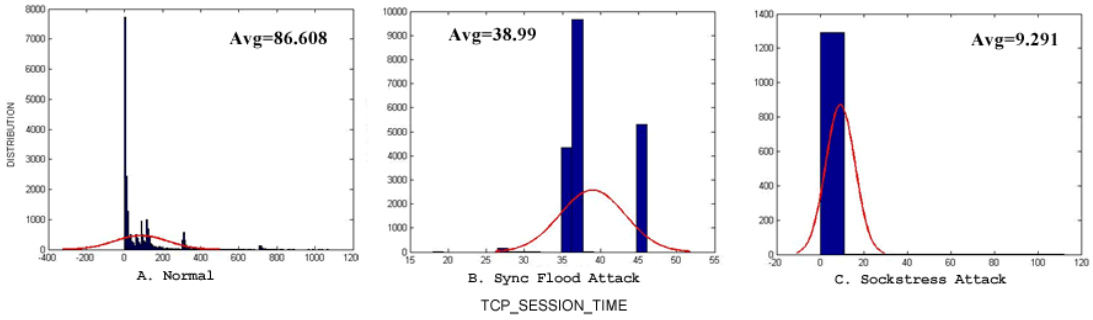


**Figure 4.17 Distribution of TCP_SESSION_TIME feature in TCP attacks**

**IS_CLIENT_FIN_TCP_CONNECTION**

This features indicate if the client close the connection normally or not. As shown Figure 4.18 both SyncFlood and SockStress attacks didn't close connection normally whereas 83% of the normal connections closed its connection normally.

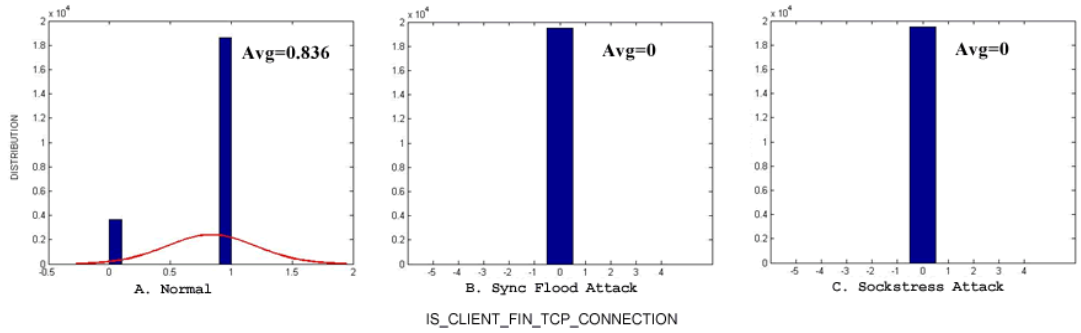


**Figure 4.18 Distribution of IS_CLIENT_FIN_ CONNECTION feature in TCP attacks**

**AVG_TCP_PAYLOAD_LENGTH**

This features holds the average size of the TCP payload packet. As shown in Figure 4.18, the average of TCP payload length in both attacks is zero byte, and this is logical because there's no data exchange before or after connection establishment. In the other hand, normal traffic average TCP payload length is 377 bytes.

**AVG_CLNT_TCP_WINDOW_SIZE**

This features show the client TCP window size, which indicate the packet size that need to be sent by the server to the client, low window size indicates high network congestion. As shown in Figure 4.19, the average window size of SockStress attack is close to zero, also it's small in SyncFlood attack, whereas in normal traffic it's extremely high compared with both attacks.

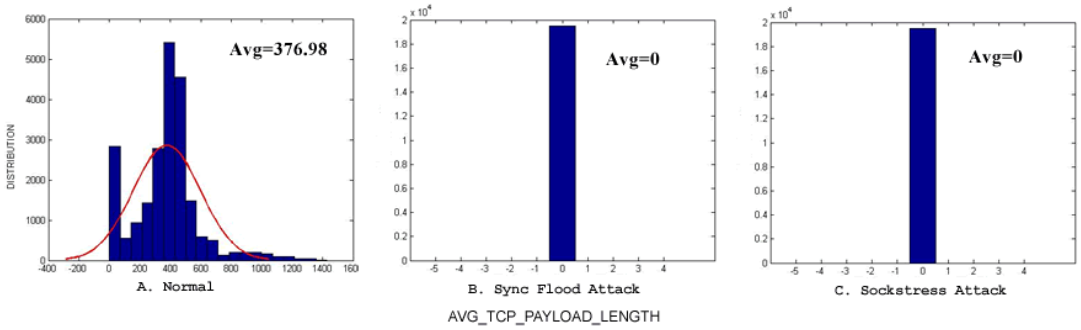


**Figure 4.18 Distribution of AVG_TCP_PAYLOA_LENGTH feature in TCP attacks**

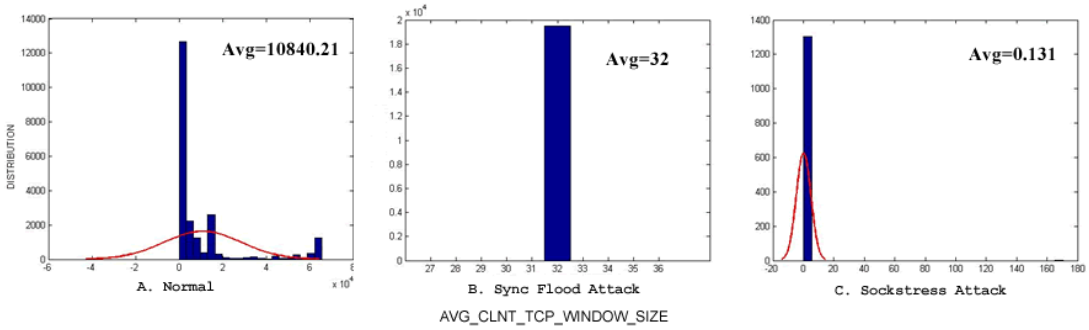


**Figure 4.19 Distribution of AVG_CLNT_WINDOW_SIZE feature in TCP attacks**

**CURRENT_CONNECTIONS_2SEC**

This features shows the number of opened connections in a time window of 2 seconds. SynFlood attack has a significant difference compared with both normal and SockStress attack where there's 14,364 opened connections in average within 2 seconds as shown in Figure 4.20 whereas in normal traffic just 8 open connections in average in a 2 seconds window time. In SockStress attack traffic the number of opened connections in a time window of 2 seconds reached 33 opened connection. For more detail about the connection's behavior of both normal and attack traffic take a look at the previous subsections 4.4.1 and 4.4.2.1 respectively.

**CURRENT_CONNECTIONS_4SEC**

This features shows the number of opened connections in a time window of 4 seconds. As shown in Figure 4.21 the number of connections is duplicated as expected compared with a time window of 2 seconds shown in Figure 4.20, but the normal traffic is not duplicated which means that the DoS attack traffic is consistent.

**NUMBER_OF_CIENT_FLOW_SYNC**

Using the same source port to send multiple SyncFlood attacks is happened only in direct SyncFlood attacks, the attack send RST message to the SYN-Acknowledgment came from the server then send a new SYN message. The average number of SYNC messages initiated using the same source port exceeds 5 in direct SyncFlood attack while it is one in both normal traffic and SockStress attack as shown in Figure 4.22.

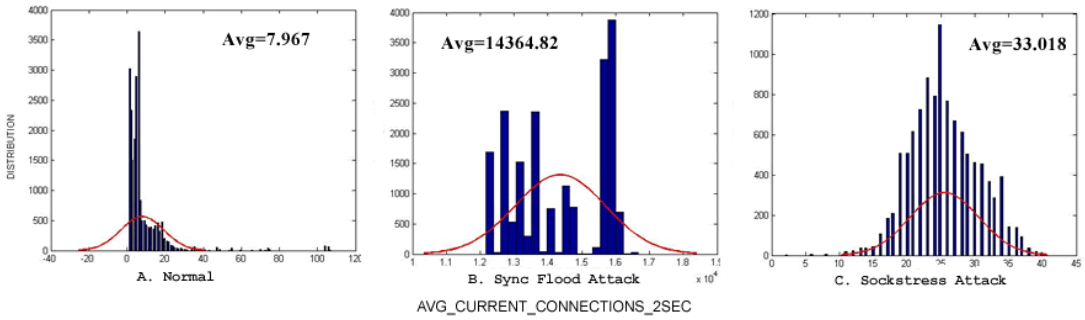


**Figure 4.20 Distribution CURRENT_CONNECTIONS_2SEC feature in TCP attacks**

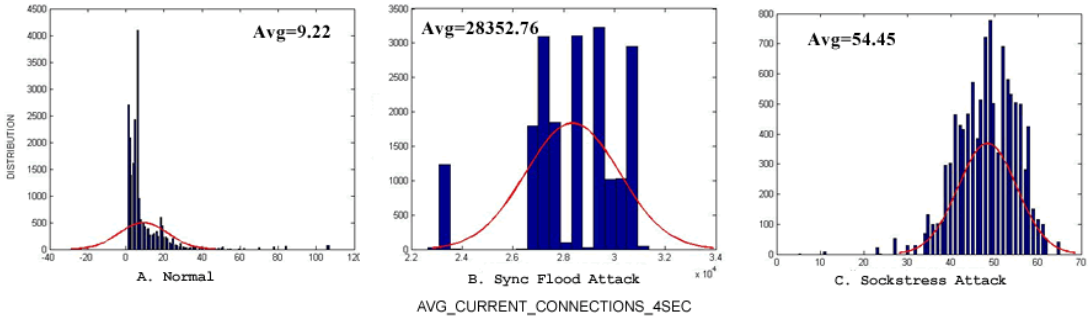


**Figure 4.21 Distribution of CURRENT_CONNECTIONS_4SEC feature in TCP attacks**

## IS_HTTP_SESSION

This feature indicates whether HTTP session is initiated or not, as shown in Figure 4.23 88% in average of the normal traffic initiated HTTP session whereas in both SynFlood and SockStress attacks the HTTP session is not initiated at all.

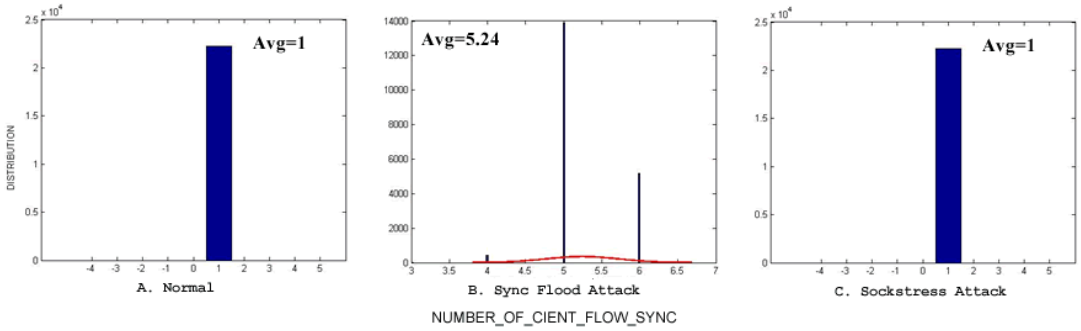


**Figure 4.22 Distribution of NUM_OF_CLIENT_FLOW_SYNC feature in TCP attacks**

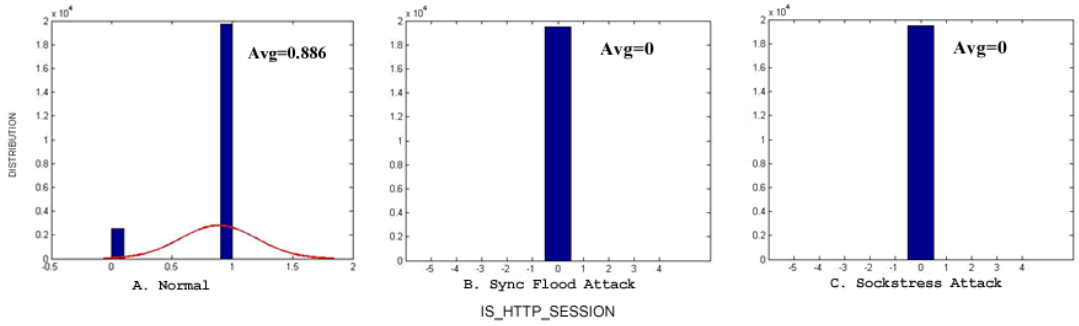


**Figure 4.23 Distribution of IS_HTTP_SESSION feature in TCP attacks**

## 4.5.1 Application Layer Attack Features

In this subsection we look deep at the differences between normal traffic and application protocol attack traffic, e.g. SlowPost attack.

### TCP_SESSION_TIME

As shown in Figure 4.24 the attack TCP session connection's time exceeds 120 seconds in average seconds which is greater than the normal connection's time which is 86.6 seconds in average. This indicates that SlowPost and SlowHeader attacks have long connection time than normal traffic, which aim to preserve the server resources as long as possible.

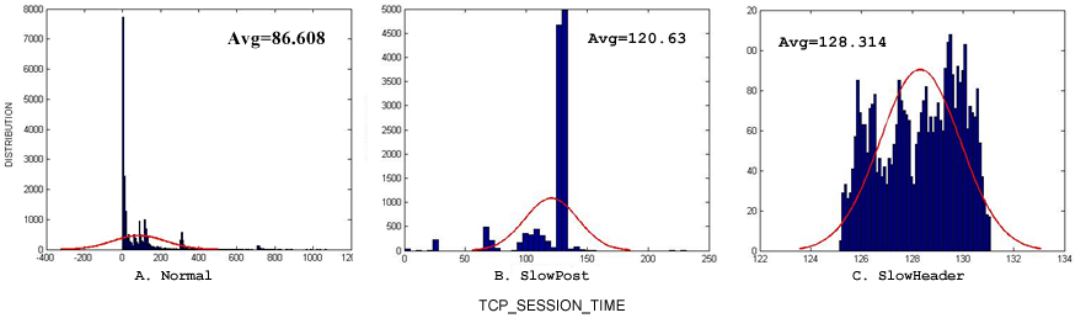


**Figure 4.24 Distribution of TCP_SESSION_TIME feature in HTTP attacks**

### HTTP_SESSION_TIME

This feature is different than TCP_SESSION_TIME which indicates the time taken to complete HTTP session which includes the client requests and server responses in the same TCP session. As shown in Figure 4.25 the HTTP session time in normal traffic doesn't exceed 33 seconds in average whereas it is higher in both SlowPost and SlowHeader attacks which exceeds 119 seconds in average. This could happen normally when uploading a big file in a slow network connection.

### IS_HTTP_HEADER_END

This feature indicates whether the HTTP header is completed or not. Figure 4.26 shows that SlowHeader attack didn't complete HTTP header which is expected as described in chapter 2.

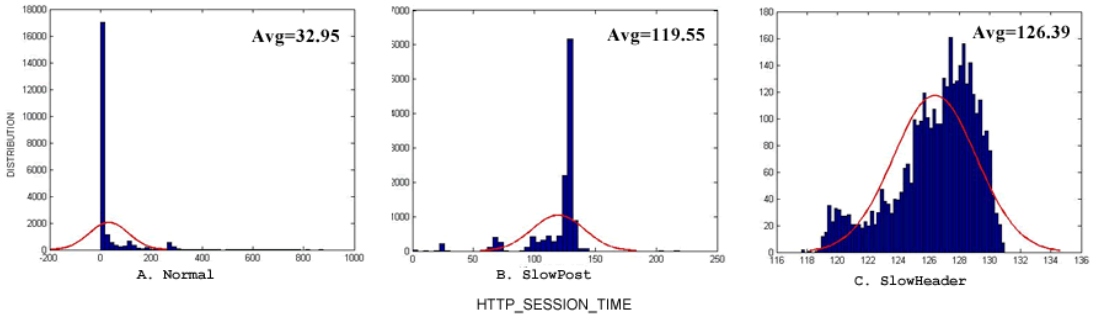


**Figure 4.25 Distribution of HTTP_SESSION_TIME feature in HTTP attacks**

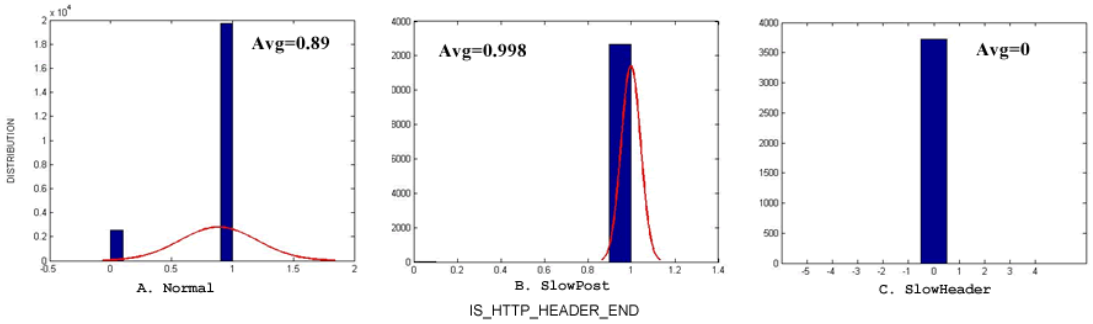


**Figure 4.26 Distribution of IS_HTTP_HEADER_END feature in HTTP attacks**

On the other hand it is close to one in both normal traffic and SlowPost traffic which indicates that the HTP header is completed. In normal traffic 89% of connections completed HTTP header where as 11% didn't send HTTP header, see feature IS_HTTP_SEESION.

### AVG_TIME_HTTP_HEADER_COMPLETE

This feature holds the time it takes while sending HTTP header which is expected to be large in SlowHeader attack as shown in Figure 4.27, sending HTTP header takes 126 seconds without complete based on Figure 4.26, whereas it takes 0 seconds to be completed in both normal traffic and SlowPost traffic which indicates that the HTTP header have been send in just one packet.

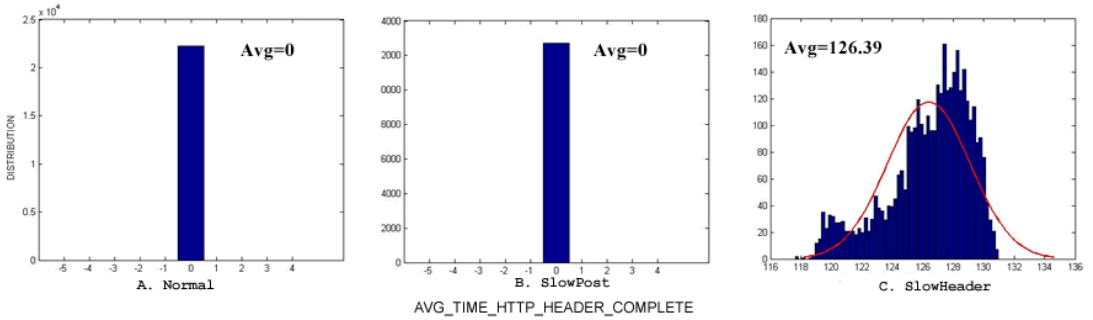


**Figure 4.27 Distribution of AVG_HTTP_HEADER_COMPLETE feature in HTTP attacks**

**NUMBER_OF_CLIENT_HTTP_HEADERS**

SlowPost and SlowHeader attacks initiate HTTP session by sending only one HTTP header as shown in Figure 4.28, but normally multiple HTTP headers is need to browse a web page where in most web sites pages there's related files like CSS and JS needed to be retrieved to view the web page as shown in normal traffic. One HTTP header request in normal connections could happen normal e.g. downloading a file.

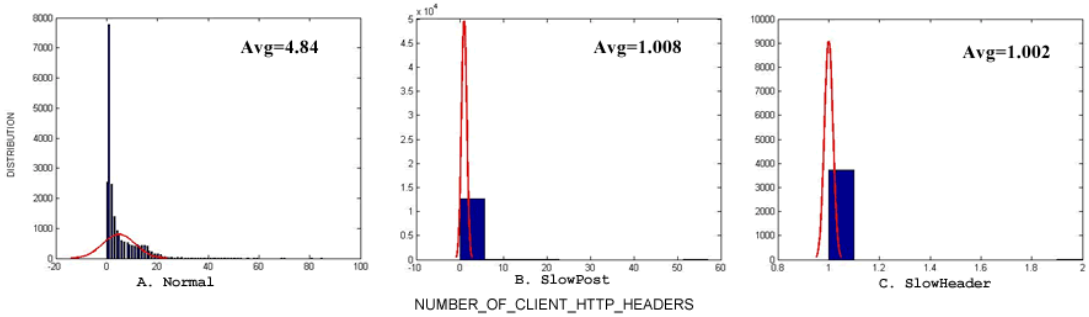


**Figure 4.28 Distribution of Number_of_Client_HTTP_Header feature in HTTP attacks**

**IS_CLIENT_FIN_TCP_CONNECTION**

Normally a TCP FIN message is sent by the client to end the TCP connection as shown in normal traffic in Figure 4.29. Whereas SlowPost and SlowHeader attack there's no TCP FIN message is sent.

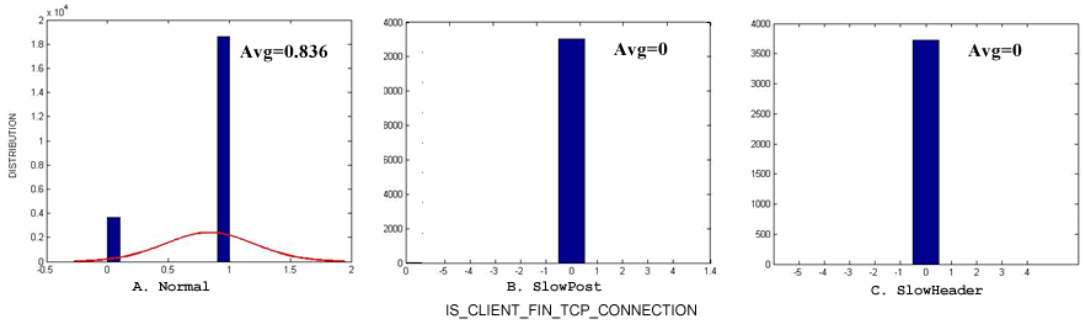


**Figure 4.29 Distribution of IS_CLNT_FIN_TCP_CONN feature in HTTP attacks**

**NUMBER_OF_CLIENT_TCP_PSH**

TCP PSH message is sent by the client to tell the server to push the TCP payload up to the application. Normally each http header is embedded in one packet which needs one TCP PSH. The number of TCP PSH in normal traffic could be large when for example uploading a large file, so there's a need to send TCP PSH message after amount number of bytes is sent to tell the server to flush the buffer and sending the data to the web application for further processing.

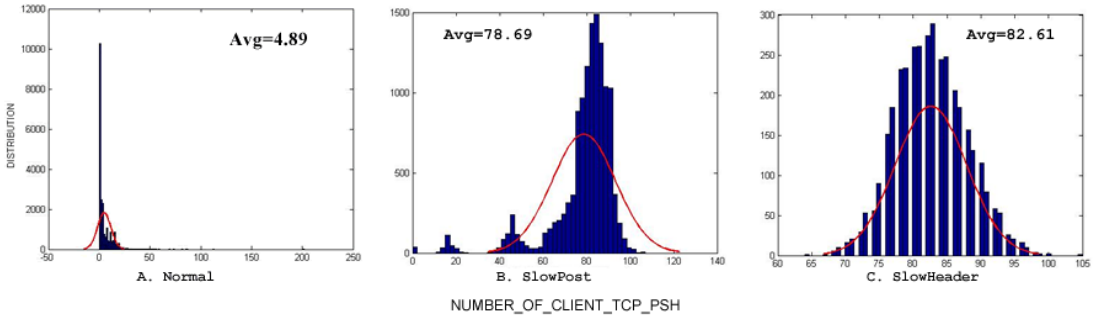


**Figure 4.30 Distribution of NUMBER_OF_CLIENT_TCP_PSH feature in HTTP attacks**

In Figure 4.30 the average number of TCP PSH messages reached 5 messages whereas there's a significant difference appeared in SlowPost and SlowHeader attack's traffic. The difference between the increase number of TCP PSH in normal and attacks traffic is given by the TCP payload length where it's very small in both SlowPost and SlowHeader attacks whereas it's large in normal traffic as shown in the next feature which is AVG_TCP_PAYLOAD length.

### AVG_TCP_PAYLOAD_LENGTH

As described in chapter 2, SlowPost and SlowHeaders send a small payload as shown in Figure 4.31, whereas in normal traffic there is significant difference**.**

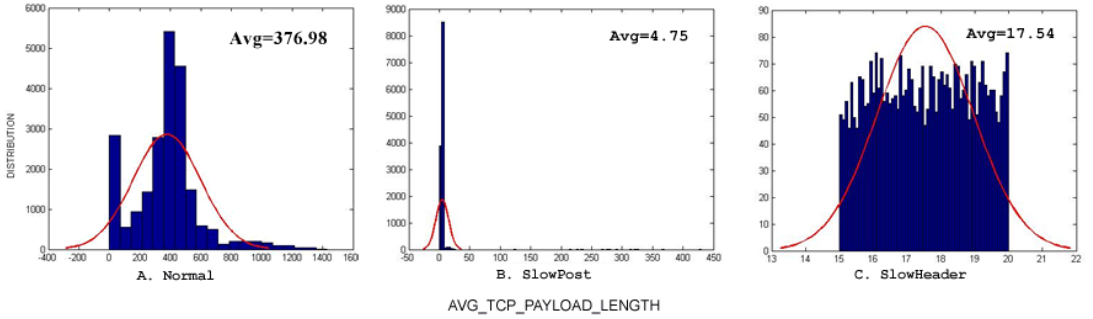


**Figure 4.31 Distribution of AVG_TCP_PAYLOAD_LENGTH feature in HTTP attacks**

### AVG_CLNT_TCP_WINDOW_SIZE

Client TCP window size as shown in Figure 4.32 is large which indicates normal behavior of both SlowHeader and SlowPost attacks.

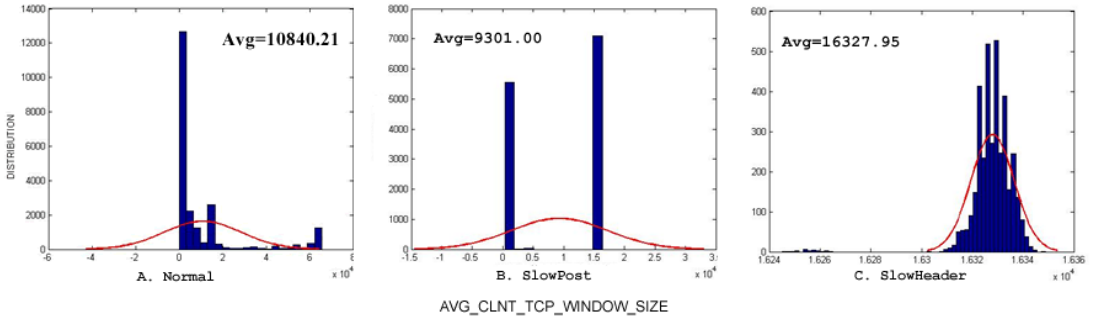


**Figure 4.32 Distribution of AVG_CLNT_TCP_WINDOW_SIZE feature in HTTP attacks**

### CURRENT_CONNECTIONS_2SEC

This features shows the number of opened connections in a time window of 2 seconds. SlowPost and SlowHeader attacks have a significant difference compared with normal traffic where there's 890 and 1580 opened connections in average respectively within 2 seconds as shown in Figure 4.33. Whereas in normal traffic just 8 open connections in average in a 2 seconds window time. For more detail about the connection's behavior of both normal and attack traffic take a look at the previous subsections 4.4.1 and 4.4.2.2 respectively.

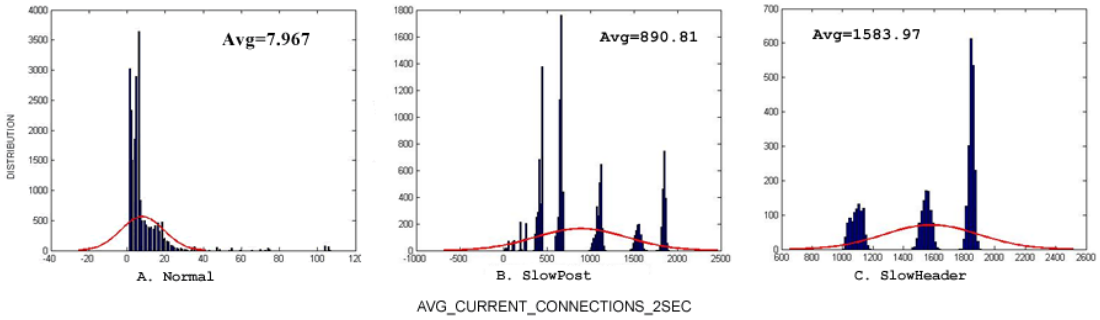


**Figure 4.33 Distribution of CURRENT_CONNECTIONS_2SEC feature in HTTP attacks**

### CURRENT_CONNECTIONS_4SEC

This features shows the number of opened connections in a time window of 4 seconds. As shown in Figure 4.34 the number of connections is not duplicated like network layer attacks, take a look at Figure 4.33, but the number of increased open connections is larger than the increase of opened connections in normal traffic. The reason of lower increase of opened connections in both attacks is due to the connection behavior of both attacks as described in subsection 4.4.2.2.

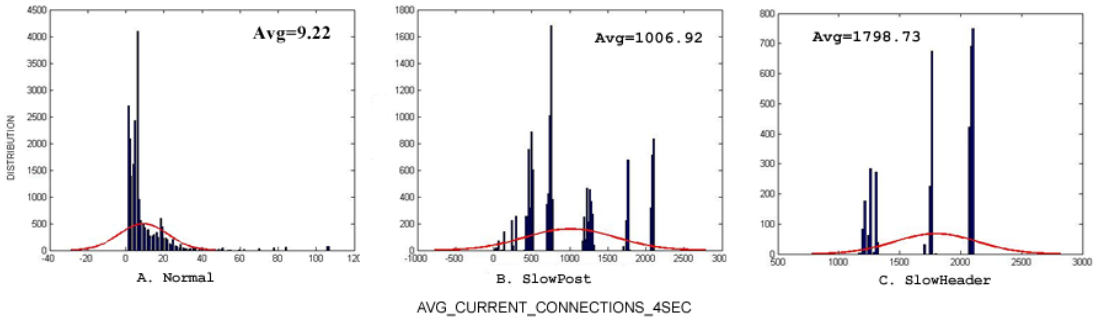


**Figure 4.34 Distribution of CURRENT_CONNECTIONS_4SEC feature in HTTP attacks**

### USER_AGENTS_2SEC

Both SlowPost and SlowHeader attacks send a random user agent to behave that the connections came from different user agents not from single agent. As shown in Figure 4.35 the number of user agents reaches 20 user agents in average while in normal traffic it is close to single user agent.

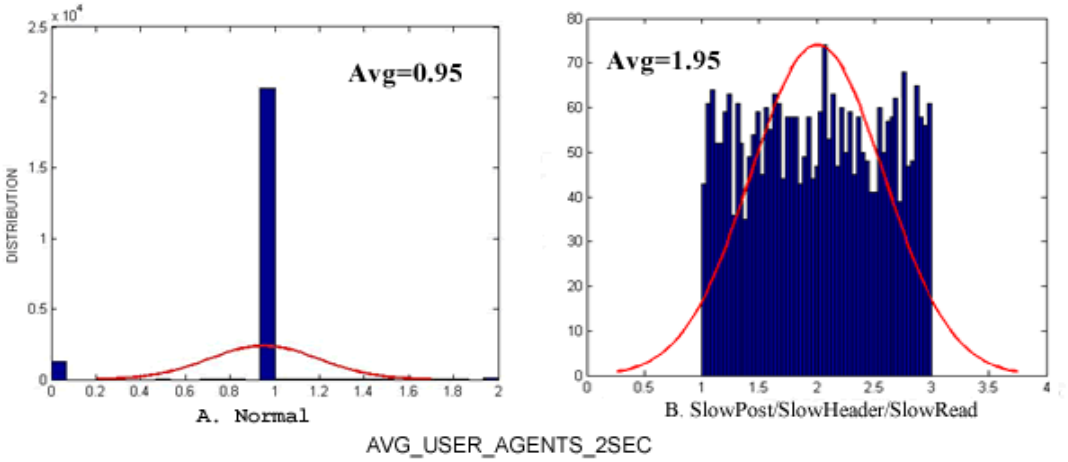


**Figure 4.35 Distribution of USER_AGENTS_2SEC feature in HTTP attacks**

**NUMBER_ZERO_WINDOW_PKTS**

This features appeared in two attacks type, SockStress attack and SlowRead attack. Compared with SockStress attack, SlowRead attack has an average of 9.75 zero window packets in a single TCP session, as shown in Figure 4.36 the number of zero window packets is almost zero in normal instances.

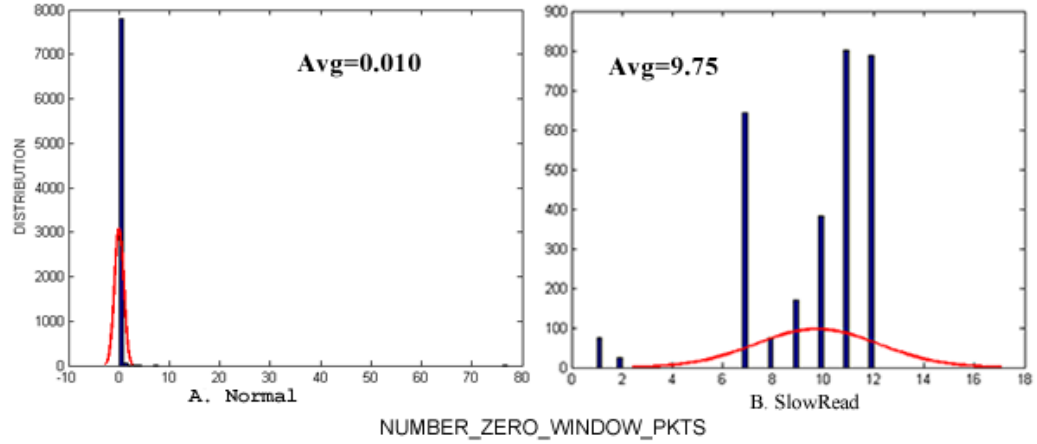


**Figure 4.36 Distribution of NUMBER_ZERO_WINDOW_PKTS feature in HTTP**

## *4.7 Summary*

Real dataset was collected from Alaqsa university network traffic which is going to and coming from its website server. The traffic packets was captured using Wireshark software which was installed on a standalone server in the Alaqsa university DMZ and was connected to a switch mirror port and the wireshark server NIC promiscuous mode was enabled.

We captured the normal traffic going to and coming from the production server while the attack traffic was captured from a website-like server which is located in an isolated network. We performed two major types of DoS attacks, which are network layer attack and application layer attack. For instance we chose SYNC-Flood and SockStres attacks for network layer attack, on the other hand SlowPost, SlowHeaders and SlowRead were chosen for application layer attack. List of features were generated and analyzed.

# Chapter 5: Research Proposal and Methodology

In this chapter, the proposed model methodology is presented and explained. The chapter organized into five sections. Section one, presents methodology steps to achieve our primary model. Section two gives a description of the collected data sets and description of their features. Section three describes the preprocessing steps and feature selection of each dataset. The fourth section contained the process of building the model including the baseline experiments to select the optimal feature sets and equations which was used for building the model. In the fifth section, the measurements of proof of concept evaluation to evaluate the performance of our model is presented.

## *5.1 Methodology Steps*

To implement and evaluate this model we need to follow the following two main phases as shown in Figure 5.2. An overview of the OCC NIDS based service's normal behavior is illustrated in Figure 5.1; each new instance is forwarded to its relevant OCC based on the service that it used.

**Training phase: The purpose of this phase is to select the optimal features set for each service and to obtain their standard deviations.**

In this phase, the optimal features set for each service is selected and normalized, the service's centroid table is built and the standard deviation of each service's normal class is obtained, as shown in Figure 5.2 (a).

**Step 1**: Download KDD Cup '99 dataset [20] which is a common used dataset. We also collected a real network traffic of HTTP service from Alaqsa University Network (BM-AUN2015), because HTTP is becoming a universal transport protocol and much data access occurs via HTTP. These reasons lead it to become a common exploit target [85, 86]. For detailed information about this dataset take a look at Chapter 4.

**Step 2:** Retrieve the normal instances from training dataset. This step needs a lot of concerns. The challenges of applying normal based detection model in networks are difficult because we can't guarantee that the existing normal activity is absolutely free from attack traces. To overcome this issue we need to apply essential steps including outlier elimination in which we will be highly guaranteed that the normal activity may have a neglected percentage of attacks. These steps is based on the assumption that most of the computers are not infected and also "The attack traffic is statistically different from normal traffic" [87, 88].

**Step 3:** Divide normal instances into subsets based on service used, each subset is used as the class of OCC model.

**Step 4:** Normalize each service's normal dataset features (e.g. packet size, duration,..) and also the testing dataset features which depends on the normalization factor of training dataset using Z-Score normalization which is discussed in data normalization in subsection 2.3.1.2.

**Step 5:** Each service's dataset is considered as class. Generate a centroid table for each class. The purpose of creating different normal classes based on service type is that each service (e.g. HTTP) has its own behavior's characteristics and features. This means that each class has its own relevant feature space which differs than other classes.

**Step 6:** Extract the most relevant features for each service, which will be used to build the model. Each service has its own features subset that differs than other services.

**Step 7:** Calculate the class's standard deviation. The standard deviation of each class is used as the class radius or the class boundary. The standard deviation of each service's class is derived by computing the distance of each instance in the service's normal dataset, from its service's class centroid using Euclidean distance. Then applying the Sample Standard Deviation to get the standard deviation of the class [16]. The equations needed for this step are shown in Eq. 5.1 and Eq. 5.2 in subsection 5.4.1.1.

**Development phase: The purpose of this phase is to calculate the model accuracy at different Tune values using a subset of features.**

**Step 1:** Retrieve a new instance from training dataset and Calculate the instance distance from its relevant service's class centroid table.

**Step 2:** Check whether the instance is normal or abnormal. The instance is labeled as abnormal if the distance is greater than the class's standard deviation, else it as labeled as normal.

**Step 3:** Calculate the overall accuracy at the selected Tune value. The tune value is a real number added to the standard deviation in order to increase the service class's boundary [16].

**Step 4:** Check if all predefined tune values are applied.

**Testing Phase: The purpose of this phase is to test the model and evaluate its testing results.**

In this phase, OCC model of each service is tested using the testing dataset of Bm-AUN2015 and KDD Cup'99 and evaluated using confusion matrix, as shown in Figure 5.2 (b).

**Step 1:** Retrieve a new instance from testing dataset, and normalize its features using the service's normal class normalization factor obtained from training phase (step 4).

**Step 2:** Calculate the new instance distance from its relevant service's class centroid table.

**Step 3:** Check whether the new instance is normal or abnormal. The instance is labeled as Abnormal if the distance is greater than the class's standard deviation, else it as labeled as Normal.

**Step 4:** Calculate the overall accuracy, detection rate, false alarm rate, and detection time..

**Step 5:** Compare our results with some previous related works.

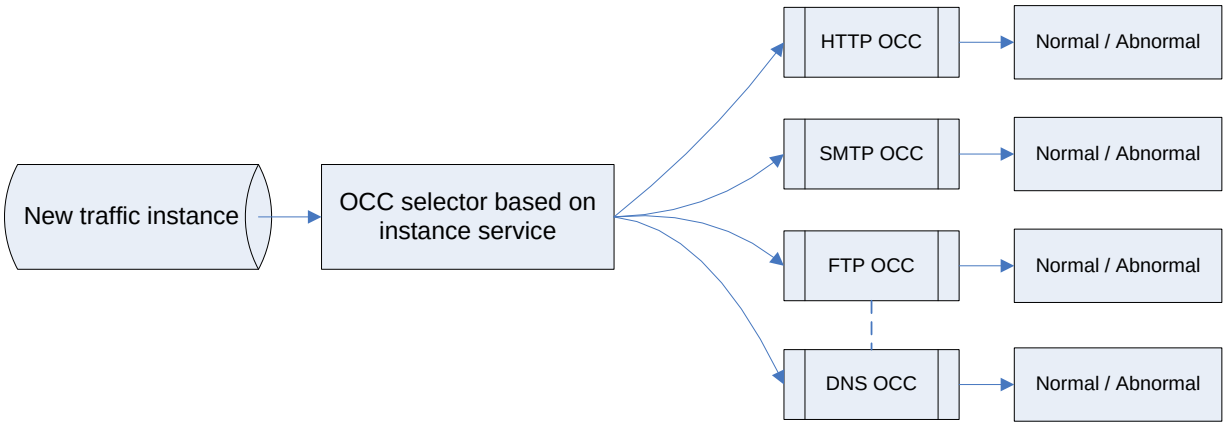


**Figure 5.1 An overview of the OCC NIDS based service's normal behavior**

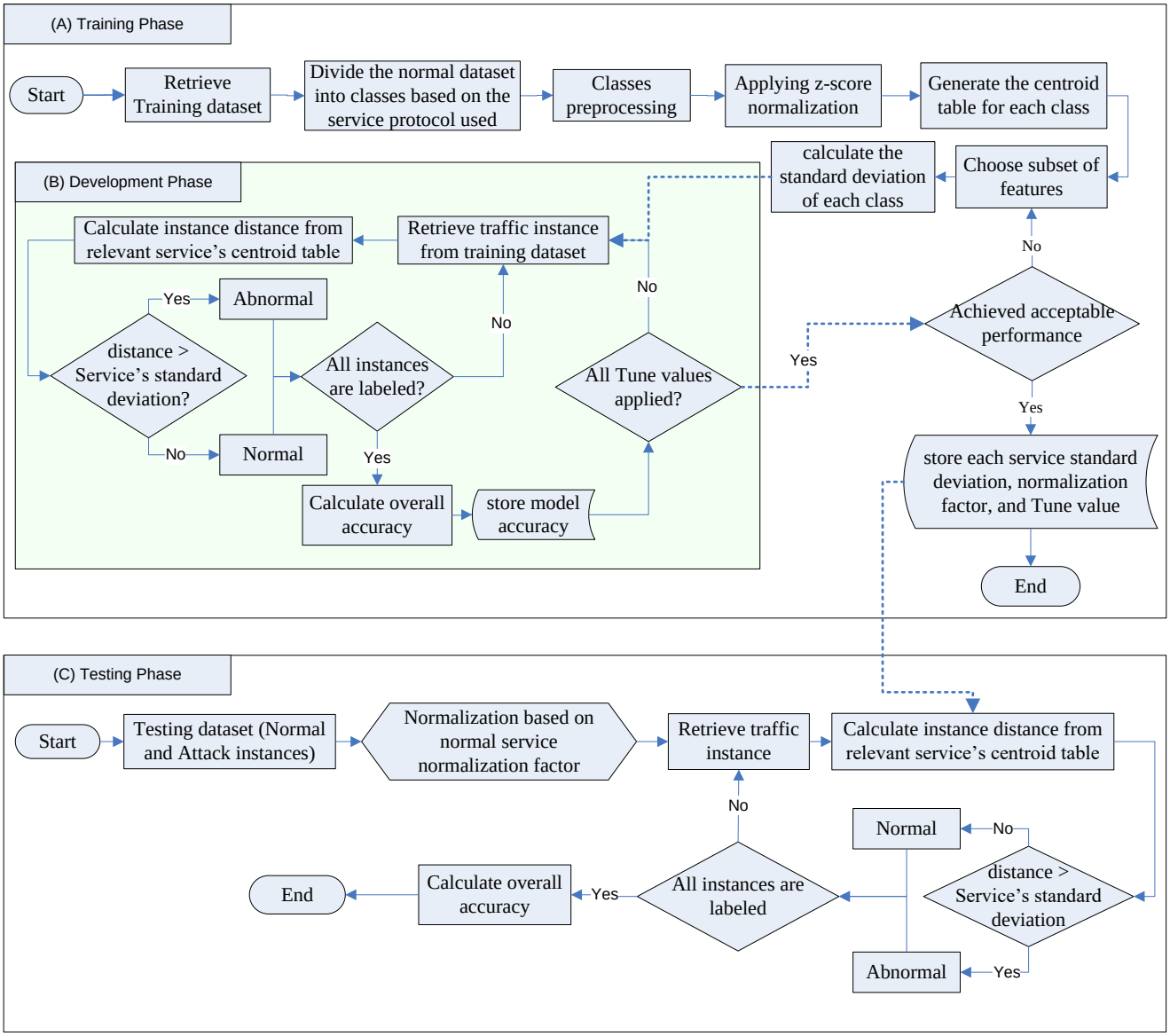


**Figure 5.2 The proposed model, (A) Training Phase, (B) Development Phase, (C) Testing Phase**

## *5.2 Datasets of Model*

In this section, data sets are presented, collected and described. The main categories, data sample and attack types are also described.

### 5.2.1 Datasets Collection:

We used two datasets to evaluate our primary model. The first dataset is a real dataset which was collected from Alaqsa University Network (BM-AUN2015), and have been described in detail in chapter 4. KDD Cup'99 [20] is the second dataset which also was used to evaluate our primary model.

The reasons of using these datasets is that, BM-AUN2015 is a real dataset which have been collected from a real traffic, and have a recent attacks that is not exists in KDD Cup'99. We consider HTTP service attacks in this dataset. Also KDD Cup'99 [20] dataset is used as a benchmarking for intrusion detection systems, and it is widely used and accepted in the academic community.

### 5.2.2 Datasets Description:

BM-AUN2015 dataset features is listed in Table 4.4 and also is described in chapter 4. The dataset contains of 19 features in addition to the class type. The number of normal and abnormal instances is shown in Table 4.5 and the training dataset instances is listed in Table 4.6.

KDD Cup'99 dataset has 41 features for each connection record plus one class label [20]. These features are grouped into four categories:

**Basic Features:** which can be derived from packet headers without inspecting the payload as shown in Table 5.1.

**Content Features:** Is a domain knowledge feature which is used to access the payload of the original TCP packets. This includes features such as number of failed login attempts as shown in Table 5.2.

**Time-based Traffic Features:** These features are designed to capture properties that mature over a 2 second temporal window. One example of such a feature would be the number of connections to the same host over the 2 second interval as shown in Table 5.3.

**Host-based Traffic Features:** Utilize a historical window estimated over the number of connections instead of time. Host-based features are designed to access attacks, which span intervals longer than 2 seconds as shown in Table 5.4.

**Table 5.1 Basic features of individual TCP connections**

| # | Feature name | Description | Data type |
|---|---|---|---|
| 1 | Duration | length (number of seconds) of the connection | Continuous |
| 2 | protocol_type | type of the protocol, e.g. tcp, udp, etc. | Polynominal |
| 3 | Service | network service on the destination, e.g., http, telnet, etc. | Polynominal |
| 4 | src_bytes | number of data bytes from source to destination | Continuous |

| | | | |
|---|---|---|---|
| 5 | dst_bytes | number of data bytes from destination to source | Continuous |
| 6 | Flag | normal or error status of the connection | Polynominal |
| 7 | Land | 1 if connection is from/to the same host/port; 0 otherwise | Binominal |
| 8 | wrong_fragment | number of wrong fragments | Continuous |
| 9 | Urgent | number of urgent packets | Continuous |

**Table 5.2 Content features within a connection suggested by domain knowledge**

| # | Feature name | Description | Data type |
|---|---|---|---|
| 10 | Hot | number of "hot" indicators | Continuous |
| 11 | num_failed_logins | number of failed login attempts | Continuous |
| 12 | logged_in | 1 if successfully logged in; 0 otherwise | Binominal |
| 13 | num_compromised | number of "compromised" conditions | Continuous |
| 14 | root_shell | 1 if root shell is obtained; 0 otherwise | Binominal |
| 15 | su_attempted | 1 if "su root" command attempted; 0 otherwise | Binominal |
| 16 | num_root | number of "root" accesses | Continuous |
| 17 | num_file_creations | number of file creation operations | Continuous |
| 18 | num_shells | number of shell prompts□ | Continuous |
| 19 | num_access_files | number of operations on access control files | Continuous |
| 20 | num_outbound_cmds | number of outbound commands in an ftp session | Continuous |
| 21 | is_hot_login | 1 if the login belongs to the ``hot'' list; 0 otherwise | Binominal |
| 22 | is_guest_login | 1 if the login is a ``guest''login; 0 otherwise | Binominal |

**Table 5.3 Traffic features computed using a two-second time window**

| # | Feature name | Description | Data type |
|---|---|---|---|
| 23 | Count | number of connections to the same host as the current connection in the past two seconds | Continuous |
| 24 | serror_rate | % of connections that have ``SYN'' errors | Continuous |
| 25 | rerror_rate | % of connections that have ``REJ'' errors | Continuous |
| 26 | same_srv_rate | % of connections to the same service | Continuous |
| 27 | diff_srv_rate | % of connections to different services | Continuous |
| 28 | srv_count | number of connections to the same service as the current connection in the past two seconds | Continuous |
| 29 | srv_serror_rate | % of connections that have ``SYN'' errors | Continuous |
| 30 | srv_rerror_rate | % of connections that have ``REJ'' errors | Continuous |
| 31 | srv_diff_host_rate | % of connections to different hosts | Continuous |

**Table 5.4 Host-based traffic features**

| # | Feature name | Description | Data type |
|---|---|---|---|
| 32 | dst host count | Count of connections having the same destination host | Continuous |
| 33 | dst host srv count | Count of connections having the same destination host and using the same service | Continuous |
| 34 | dst host same srv rate | % of connections having the same destination host and using the same service | Continuous |
| 35 | dst host diff srv rate | % of different services on the current host | Continuous |
| 36 | dst host same src port rate | % of connections to the current host having the same src port | Continuous |
| 37 | dst host srv diff hostrate | % of connections to the same service coming from different hosts | Continuous |
| 38 | dst host serror rate | % of connections to the current host that have an S0 error | Continuous |
| 39 | dst host srv serror rate | % of connections to the current host and specified service that have an S0 error | Continuous |
| 40 | dst host rerror rate | % of connections to the current host that have an RST error | continuous |
| 41 | dst host srv rerror rate | % of connections to the current host and specified service that have an RST error | Continuous |

### 5.2.3 Datasets Sample:

Sample of data sets was chosen randomly as shown in Table 5.5 and Table 5.6 for BM-AUN2015 and KDD cup'99 datasets respectively. We chose some features because of limited space. For more information about BM-AUN2015 full features generated and its description see Table 4.4.

**Table 5.5 BM-AUN2015 dataset sample**

| CLIENT_REPLY_ACK | IS_HTTP_SESSION | IS_HTTP_HEADER_END | AVG_TIME_HTTP_HEADER_COMPLETE | NUMBER_OF_CLIENT_TCP_PSH | AVG_TCP_PAYLOAD_LENGTH | AVG_CLNT_TCP_WINDOW_SIZE | AVG_CURRENT_CONNECTIONS_2SEC | NUMBER_ZERO_WINDOW_SIZE | CLASS_TYPE |
|---|---|---|---|---|---|---|---|---|---|
| Y | Y | Y | 0 | 18 | 418.6 | 62672.76 | 4 | 0 | Normal |
| Y | Y | Y | 0 | 10 | 377.47 | 64224.06 | 20 | 0 | Normal |
| Y | Y | Y | 0 | 3 | 25.75 | 458.41 | 1 | 0 | Normal |
| N | N | N | 0 | 0 | 0 | 0 | 1 | 0 | SYN-FLOOD-RandomIP |
| N | N | N | 0 | 0 | 0 | 0 | 15639 | 0 | SYN-FLOOD-SingleIP |
| Y | N | N | 0 | 0 | 0 | 0 | 27 | 1 | SockStress20Thread |
| Y | N | N | 0 | 0 | 0 | 0 | 44 | 1 | SockStress40Thread |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Y | Y | Y | 0 | 87 | 3.42 | 16333.52 | 416 | 0 | SlowPost |
| Y | Y | N | 124.4 | 83 | 18.51 | 16329.26 | 998 | 0 | SlowHeader |
| Y | Y | Y | 0 | 0 | 23.5 | 355 | 250 | 5 | SlowRead |

**Table 5.6 KDD Cup'99 dataset sample**

| Duration | Protocol_type | Service | Flag | Src_bytes | Dst_bytes | type |
|---|---|---|---|---|---|---|
| 0 | Tcp | http | SF | 181 | 5450 | normal. |
| 0 | Tcp | http | SF | 217 | 2032 | normal. |
| 0 | Icmp | ecr_i | SF | 1932 | 0 | smurf. |
| 0 | Tcp | Private | S0 | 0 | 0 | neptune. |
| 2 | Tcp | Smtp | SF | 1572 | 437 | normal. |
| 2 | Udp | Domain_u | SF | 93 | 37 | normal. |

## *5.3 Preprocessing Datasets and Features Selection:*

In this section preprocessing data sets, outlier removing, normalization and features selection have been explained.

### 5.3.1 Datasets Preprocessing:

Preprocessing of dataset is necessary step to make it a suitable input for the classification process. The nominal/symbolic features have been converted to binary values which is suitable for distance measurements, also outliers have been removed from the normal dataset which is needed to build the model.

**Outlier Detection and Remove**

In order to remove outliers from the normal dataset we used the Local Outlier Probability (LoOP) [89] to detect the abnormal instances in the normal dataset and eliminate all the instances which have an outlier probability greater than 0.7.

LoOP, which is proposed by Kriegel et al [89], is a method derived from Local Outlier Factor (LOF) but using inexpensive local statistics to become less sensitive to the choice of the parameter k, in addition, the resulting values are scaled to a value range of [0:1] which calculates the outlier score based on Local Outlier Probability.

**Z-Score Normalization**

By using this normalization method, the values for an attribute A are normalized based on mean and standard deviation of A. This method is used when the actual minimum and maximum of an attribute is unknown , and to avoid the outliers that dominate the min-max normalization. Z-score normalization equation is shown in Eq. 2.3 in subsection 2.3.1.2.

We used Z-score normalization because the maximum values of some features like packet payload is not bounded and also some network features such as packet payload, TCP window

size, and number of concurrent connections in 4 sec may have big number compared with other features such as number of different user agents used in 4 sec.

### *5.3.2.1 Preprocessing of BM-AUN2015 Dataset*

For more information about BM-AUN2015 dataset features and their description please refer to Chapter 4 Table 4.4.

**Conversion of Nominal Features:**

As shown in Table 4.4, BM-AUN2015 dataset contains 4 binominal features which needed to be converted to numerical features in order to be used with numerical distance measurement. These features are CLIENT_REPLY_ACK, IS_HTTP_SESSION, IS_HTTP_HEADER_END, IS_CLIENT_FIN_TCP_CONNECTION. Table 5.7 shows BM-AUN2015 dataset features after conversion.

**Table 5.7 BM-AUN2015 dataset with nominal features converted to numerical**

| # | Feature Name | Type | Description |
|---|---|---|---|
| 1 | FLOWKEY | Integer | The TCP Session ID. |
| 2 | CLIENT_REPLY_ACK_TRUE | Integer | Client complete the handshaking. |
| 3 | CLIENT_REPLY_ACK_FALSE | Integer | Client don't complete the handshaking. |
| 4 | CLIENT_TIME_TO_REPLY_ACK | Integer | The time after the last Server SYNC ACK. |
| 5 | NUMBER_OF_SERVER_ACK | Integer | Number of Server Ack to Sync message. |
| 6 | TCP_SESSION_TIME | Integer | The total time of the TCP session. |
| 7 | HTTP_SESSION_TIME | Integer | The total time of the HTTP session. |
| 8 | IS_HTTP_SESSION_TRUE | Integer | this session include an HTP session. |
| 9 | IS_HTTP_SESSION_FALSE | Integer | this session not include an HTTP session. |
| 10 | IS_HTTP_HEADER_END_TRUE | Integer | HTTP Request headers is ended. |
| 11 | IS_HTTP_HEADER_END_FALSE | Integer | HTTP Request headers isn't ended. |
| 12 | AVG_TIME_HTTP_HEADER_COMPLETE | Real | Average time to complete HTTP header. |
| 13 | NUMBER_OF_CLIENT_HTTP_HEADERS | Integer | # headers sent in the same TCP session. |
| 14 | IS_CLIENT_FIN_TCP_CONNECTION_TRUE | Bolean | the client end the TCP session. |
| 15 | IS_CLIENT_FIN_TCP_CONNECTION_FALSE | Bolean | the client didn't end the TCP session. |
| 16 | NUMBER_OF_CLIENT_TCP_PSH | Integer | Total # of client packets with PSHflag set. |
| 17 | AVG_TCP_PAYLOAD_LENGTH | Real | The average length of TCP packet payload. |
| 18 | AVG_CLNT_TCP_WINDOW_SIZE | Real | The average size of the client TCP window. |
| 19 | CURRENT_CONNECTIONS_2SEC | Integer | # of connections 2 sec time window. |

| | | | |
|---|---|---|---|
| 20 | CURRENT_CONNECTIONS_4SEC | Integer | # of connections 4 sec time window. |
| 21 | USER_AGENTS_2SEC | Integer | # of distinct user agents used in 2 sec. |
| 22 | NUMBER_OF_CIENT_FLOW_SYNC | Integer | # of SYNC sent in current TCP session. |
| 23 | NUMBER_ZERO_WINDOW_PKTS | Integer | # of client zero window size packets |
| 24 | CLASS_TYPE | String | Instance class. |

**Features Selection**

We chose Weight by Information Gain Ratio operator in RapidMiner, which calculates the weight of attributes with respect to the label attribute by using the information gain ratio. The higher the weight of an attribute, the more relevant it is considered as shown in Table 5.8. Based on Table 5.8 we try to find the optimal feature set in the following subsection 5.4.

**Table 5.8 BM-AUN2015 features' information gain ration**

| # | Feature Name | Dependency |
|---|---|---|
| 1 | IS_CLIENT_FIN_TCP_CONNECTION | 0.73087585 |
| 2 | NUMBER_OF_CIENT_FLOW_SYNC | 0.83498221 |
| 3 | IS_HTTP_HEADER_END | 0.84154597 |
| 4 | HTTP_SESSION_TIME | 0.85990159 |
| 5 | IS_HTTP_SESSION | 0.86017857 |
| 6 | NUMBER_OF_CLIENT_HTTP_HEADERS | 0.86589317 |
| 7 | AVG_TCP_PAYLOAD_LENGTH | 0.86589317 |
| 8 | TCP_SESSION_TIME | 0.88021912 |
| 9 | CLIENT_REPLY_ACK | 0.98938716 |
| 10 | CLIENT_TIME_TO_REPLY_ACK | 0.98938716 |
| 11 | NUMBER_OF_CLIENT_TCP_PSH | 0.99464652 |
| 12 | NUMBER_OF_SERVER_ACK | 0.99759336 |
| 13 | AVG_CLNT_TCP_WINDOW_SIZE | 0.9996767 |
| 14 | NUMBER_ZERO_WINDOW_PKTS | 0.9999731 |
| 15 | AVG_TIME_HTTP_HEADER_COMPLETE | 1 |
| 16 | AVG_CURRENT_CONNECTIONS_2SEC | 1 |
| 17 | AVG_CURRENT_CONNECTIONS_4SEC | 1 |
| 18 | AVG_USER_AGENTS_2SEC | 1 |

### *5.3.2.2 Preprocessing KDD Cup'99 dataset*

For more information about KDD Cup '99 dataset features and their description please refer to subsection 5.2.2.

**Conversion of Nominal Features:**

As shown in Table 5.1 and Table 5.2, KDD Cup'99 dataset contains 9 discrete features which needed to be converted to numerical features in order to be used with numerical distance measurement. Three features are polynominal and the others are binominal. These features

are protocol_type, service, flag, land, logged_in, root_shell, su_attempted, is_hot_login, is_guest_login. In our model, the service and protocol_type feature will be eliminated because the model is based on the service type which operates on a specific protocol type.

**Features Selection**

Features selection was chosen based on Rough Set which has been performed by Olusola et-al in [90]. They presented the relevance of each feature in KDD '99 intrusion detection dataset to the detection of each class. Rough set degree of dependency and dependency ratio of each class were employed to determine the most discriminating features for each class.

Rough Set [91] is a useful mathematical tool to deal with imprecise and insufficient knowledge, reduce data sets size, find hidden patterns and generate decision rules. Rough set theory contributes immensely to the concept of reducts. Reducts is the minimal subsets of attributes with most predictive outcome. Rough sets are very effective in removing redundant features from discrete data sets. Rough set concept is based on a pair of conventional sets called lower and upper approximations. The lower approximation is a description of objects which are known in certainty to belong to the subject of interest, while upper approximation is a description of objects which possibly belong to the subset.

The training set employed for the analysis was the "10% KDD" dataset. Since the degree of dependency was calculated for features based on entropy, redundant records from the dataset were removed since rough set does not require duplicate instances to classify and identify discrimination [90]. They listed features for which the class is selected most relevant for every type of attack and normal case. We chose three services, which are ECR_I, HTTP and POP3. These services have an enough normal instances for training and testing and also the includes most of the attack types and most sensitive services used by most users. Based on the selected features results in [90] we tried to find the most relevant features for each of the selected services that we chose. Note that the Probe attacks have a few instances per service which may not be effective in the overall model accuracy, so we need to increase these instances, we included all the probe and neptune attacks that exploited TCP protocol into HTTP and POP3 datasets, while those that exploited ICMP protocol was included into ECR_I dataset.

***HTTP service***

The number of normal instances of HTTP service exists in the 10% training dataset is 61,886 instances, where as the number of normal instances of HTTP service exists in the 10% testing dataset is 39,247 instances. Table 5.9 shows the attack types that exploit this service, their main category and the number of examples of each type.

**Table 5.9 Attack types exists in KDD Cup'99 HTTP Service**

| 10% training dataset has 61,886 normal instances | | | 10% testing dataset has 39,247 normal instances | | |
|---|---|---|---|---|---|
| **Attack** | **Count** | **Type** | **Attack** | **Count** | **Type** |
| satan. | 1416 | Probe | apache2. | 794 | DoS |
| neptune. | 107,201 | DoS | neptune. | 58001 | DoS |
| portsweep. | 1039 | Probe | portsweep. | 354 | Prope |
| phf. | 4 | R2L | phf. | 2 | R2L |
| ipsweep. | 94 | Probe | saint. | 607 | Prope |
| back. | 2203 | DoS | back. | 1098 | DoS |

### *ECR_I service*

The number of normal instances of ECR_I service exists in the 10% training dataset is 345 instances, where as the number of normal instances of ECR_I service exists in the 10% testing dataset is 173 instances.

**Table 5.10 Attack types exploit KDD Cup'99 ECR_I Service**

| 10% training dataset has 345 normal instances | | | 10% testing dataset has 173 normal instances | | | Full dataset has 3,456 normal instances | | |
|---|---|---|---|---|---|---|---|---|
| **Attack** | **Count** | **Type** | **Attack** | **Count** | **Type** | **Attack** | **Count** | **Type** |
| ipsweep. | 1153 | Probe | ipsweep. | 306 | Probe | ipsweep. | 11557 | Probe |
| pod. | 259 | DoS | pod. | 81 | DoS | pod. | 259 | DoS |
| portsweep. | 1 | Probe | saint. | 102 | Probe | portsweep. | 6 | Probe |
| smurf. | 280790 | DoS | smurf. | 164091 | DoS | smurf. | 2807886 | DoS |
| nmap. | 103 | Probe | | | | nmap | 1032 | Probe |
| | | | | | | satan. | 37 | Probe |

Table 5.10 shows the attack types that exploit this service, their main category and the number of examples of each type.

### *POP3 service*

The number of normal instances of POP3 service exists in the 10% training dataset is 79 instances, while the number of normal instances of POP3 service exists in the 10% testing dataset is 15 instances. Table 5.11 shows the attack types that exploit this service, their main category and the number of examples of each type.

**Table 5.11 Attack types exploit KDD Cup'99 POP3 Service**

| 10% training dataset has 79 normal instances | | | 10% testing dataset has 15 normal instances | | |
|---|---|---|---|---|---|
| **Attack** | **Count** | **Type** | **Attack** | **Count** | **Type** |
| neptune. | 107201 | DoS | neptune. | 58001 | DoS |

| | | | | | |
|---|---|---|---|---|---|
| portsweep. | 1039 | Probe | guess_pass. | 3642 | R2L |
| satan. | 1416 | Probe | mscan. | 1053 | Probe |
| nmap. | 231 | Probe | | | |

## *5.4 Design and Building the Model*

In this section, the base line experiments are discussed to design and build the model. Two main stages which are required to build the model are discussed. The first stage is determining the OCC feature set, the second stage is calculating the standard deviation of the service's normal instances and choose the most accurate deviation from normal that has lower false positive rate and higher detection rate. In this work the positive class is the normal class while the negative class is the attack class.

### 5.4.1 The Base Line Experiment

These experiments were performed on BM-AUN2015 and KDD Cup'99 datasets separately. The main goal from these experiments is to determine the most relevant features set and the most accurate deviation from the normal standard deviation which is calculated based on the selected feature of each experiment.

#### *5.4.1.1 Standard Deviation of Normal Service's Class Calculation*

Before performing this stage we need to perform the data preprocessing as described in section 5.3.1. The standard deviation, as described on chapter 2, is used to measure the normal distribution of the instances, the normal instances in our case. The standard deviation of each service class is derived by computing the distance of each instance in the service's normal dataset from the service normal class's centroid using the Euclidean distance equation shown in Eq 2.5, shown in subsection 2.3.1.2, then applying the Sample Standard Deviation to get the standard deviation of the service class as shown in the general equation Eq. 1.1, in subsection 2.3.1.2 The Eucliden distance is calculated using the following formula as shown in Eq. 5.1:

$$Xdist = \sqrt[2]{\sum_{i=1}^{F}(Xi - Ci)^2}$$ ---------- (Eq. 5.1) [38]

Where Xdist is the distance of instance X from the service class's centroid C, Xi is the feature i of the instance, Ci is the feature i in the class centroid and F is the total number of features of the instance based on its protocol type. After calculating the distances of all the instances from the desired class based on the service type, we use Eq.5.2 to get the sample standard deviation of the service normal class, which is used as the boundary or radius of the class.

$$s = \sqrt[2]{\frac{1}{N-1}\sum_{i=1}^{N}(Xdist_i)^2}$$ ---------- (Eq. 5.2) [16]

Where N is the number of all service's normal instances. We have created a function which return the standard deviation of the normal class. This function is listed in Appendix A.

#### *5.4.1.2 Extracting the Most Relevant Service's Feature Set*

In the following subsections we discuss the experiments that we conducted on BM-AUN2015 and KDD Cup'99 datasets in order to select the most relevant feature set of the service class. Also to find the most suitable deviation value from the service's standard deviation to achieve a lower false positive rate and higher detection rate.

### *BM-AUN2015 Dataset's Features Extraction and Model Building*

Based on the information gain ratio of BM-AUN2015 dataset, as shown Table 5.8, we performed 15 experiments, each experiment was performed on different combination of features, and each experiment also was applied using 17 different tune value. The tune value is a real number added to the standard deviation in order to increase the service class's boundary [16].

We chose Day3, shown in Table 4.1, as the service normal class and measured its standard deviation using Eq.5.2. The train was done on three types of application layer attacks which are SlowHeader-Senario1, SlowPost- Senario1 and SlowRead- Senario1 and on Day3 normal instances. For more information about the settings of these attacks please refer to subsection 4.2.1.2. The training dataset used in feature selection and base line experiment is shown in Table 4.6.

**Table 5.12 BM-AUN2015 OCC feature selection experiments' accuracy results**

| **The standard deviation of this HTTP OCC model built on experiment 15 is 2.698.** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TUNE** | **Experiment Id** | | | | | | | | | | | | | | |
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **0** | 88.3 | 88.3 | 83.5 | 82.6 | 80.5 | 80.3 | 81.6 | 87 | 80.5 | 81.9 | 81.4 | 84.9 | 84.8 | 85.1 | **85.6** |
| **1** | 80.6 | 80.1 | 80.1 | 79.2 | 90.8 | 90.4 | 93.1 | 83.9 | 91 | 92.2 | 92.6 | 94.1 | 92.1 | 94.6 | **92.6** |
| **2** | 84.3 | 85 | 85.4 | 85.4 | 95.6 | 95.9 | 96.9 | 85.3 | 96.9 | 97.7 | 97.7 | 97.6 | 96.7 | 97.7 | **97.7** |
| **3** | 82.6 | 82.6 | 82.6 | 82.8 | 93.5 | 94.5 | 97.7 | 84 | 97 | 98.4 | 98.4 | 98.1 | 98 | 98.3 | **98.4** |
| **4** | 80.6 | 80.8 | 80.8 | 80.7 | 90.8 | 90 | 98 | 81.2 | 95 | 98.5 | 98.7 | 98.1 | 98.5 | 98.4 | **98.9** |
| **5** | 76.6 | 76.7 | 76.7 | 76.7 | 86.8 | 86.4 | 97.9 | 76.9 | 94.5 | 98.4 | 98.4 | 98 | 98.5 | 98.1 | **98.9** |
| **6** | 74.1 | 74.2 | 74.2 | 74.1 | 84.5 | 84.5 | 98 | 74.4 | 93.2 | 98.2 | 98.2 | 97.8 | 98.5 | 98.1 | **98.9** |
| **7** | 73.6 | 73.6 | 73.6 | 73.6 | 84.3 | 84.4 | 98.1 | 73.9 | 91.5 | 98.1 | 98.1 | 97.8 | 98.7 | 98.1 | **98.9** |
| **8** | 73.7 | 73.7 | 73.7 | 73.7 | 83.8 | 83.8 | 97.4 | 73.8 | 87.7 | 98.5 | 98.5 | 97.5 | 98.9 | 97.4 | **99.9** |
| **9** | 73.7 | 73.7 | 73.7 | 73.7 | 83.1 | 82.8 | 96.4 | 73.7 | 84.2 | 97.8 | 97.8 | 96.8 | 98.9 | 96.4 | **99.9** |
| **10** | 73.6 | 73.6 | 73.6 | 73.6 | 81.2 | 80.2 | 93.4 | 73.6 | 81.7 | 96.9 | 96.9 | 96 | 99 | 93.4 | **99.9** |
| **12** | 74.3 | 74.3 | 74.3 | 74.3 | 75.2 | 74.9 | 88.8 | 74.3 | 75.3 | 94.4 | 94.4 | 93.5 | 99 | 88.8 | **99.9** |
| **14** | 73.7 | 73.7 | 73.7 | 73.4 | 73.6 | 73.5 | 88.5 | 73.6 | 74.6 | 94.1 | 94.1 | 94.2 | **99.9** | 88.4 | **99.8** |
| **16** | 72.5 | 72.5 | 72.5 | 72.2 | 72.3 | 72.3 | 88.4 | 72.3 | 74.5 | 94.1 | 94.1 | 94.2 | **99.9** | 88.3 | **99.8** |
| **20** | 71.8 | 71.8 | 71.8 | 71.8 | 71.9 | 71.9 | 88.2 | 71.8 | 72.4 | 94 | 94 | 94.1 | 99.8 | 88.2 | **99.8** |
| **24** | 71.6 | 71.6 | 71.6 | 71.6 | 71.7 | 71.7 | 87.8 | 71.6 | 71.9 | 93.8 | 93.8 | 93.8 | 99.6 | 87.8 | **99.6** |
| **26** | 71.5 | 71.5 | 71.5 | 71.5 | 71.6 | 71.6 | 87.7 | 71.5 | 71.9 | 93.8 | 93.8 | 93.8 | 99.6 | 87.6 | **99.6** |
| **30** | 71.3 | 71.3 | 71.3 | 71.2 | 71.4 | 71.4 | 87.4 | 71.2 | 71.8 | 93.6 | 93.6 | 93.8 | 99.6 | 87.4 | **99.5** |

As shown in the experiments results in Table 5.12, the fifteen and thirteen experiments appeared to have the highest accuracy rate 99.9% at the tune values 8,9,10 and 12 in experiment 15 and tune values 14,16 in experiment 13. We chose experiment 15 because the number of features used is 11 features which are less than experiment 13 as listed in Table 5.13, whereas experiment 13 has 12 features used. Also another reason is that experiment 15 appears to be more robust than experiment 13, the accuracy of experiment 15 didn't decrease rapidly as the tune value increase compared with experiment 13.

The results of this experiment, shown in Table 5.14, show that the selected features in Table 5.13 achieved high accuracy rate. Where the detection accuracy of the normal instances reaches 99.9% and the detection accuracy of attack instances stay 100% without any decrease as the tune value increase. We show the tune values at which there were changes in detection rates.

**Table 5.13 BM-AUN2015 OCC selected feature set**

| # | Feature Name | # | Feature Name |
|---|---|---|---|
| 1 | CLIENT_REPLY_ACK | 6 | AVG_TCP_PAYLOAD_LENGTH |
| 2 | NUMBER_OF_SERVER_ACK | 7 | AVG_CLNT_TCP_WINDOW_SIZE |
| 3 | IS_HTTP_SESSION | 8 | CURRENT_CONNECTIONS_4SEC |
| 4 | IS_HTTP_HEADER_END | 9 | USER_AGENTS_2SEC |
| 5 | NUMBER_OF_CLIENT_TCP_PSH | 10 | NUMBER_ZERO_WINDOW_PKTS |

The plot of Table 5.14 is shown in Figure 5.3, as shown, the accuracy of the normal class is increase whenever the tune value is increase, which means that the attack instances are far away from the normal class boundary.

**Table 5.14 BM-AUN2015 base line experiment results**

| | Tune value | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | **0** | **1** | **2** | **3** | **4** | **5** | **9** | **10** | **14** | **16** | **30** |
| Normal | 0.712 | 0.853 | 0.954 | 0.969 | 0.979 | 0.98 | 0.998 | 0.999 | 0.9992 | 0.9994 | 0.9996 |
| SlowHeader-Scenario1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| SlowPost-Scenario1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| SlowRead-Scenario1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

As shown in Figure 5.3 the detection accuracy of normal instances is increased in every expanding value of its standard deviation using the tune parameter. Table 5.14 shows that the detection accuracy of normal instances was 71.2% without any expanding value of its normal class standard deviation, which means that 71.2% of normal instances fall within the standard deviation and the other instances outer this boundary are relatively have an extreme values

which need to be included in the normal class boundary by increasing the tune value parameter.

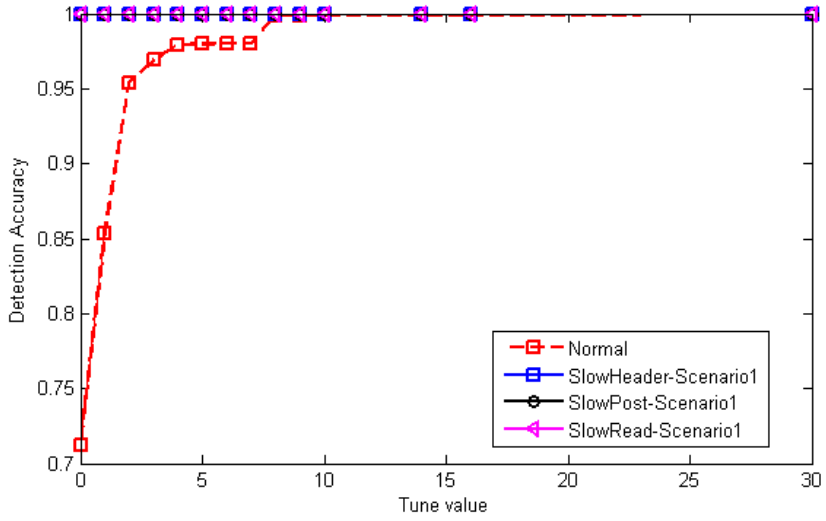


**Figure 5.3 BM-AUN2015 base line experiment result chart**

On the other hand, attack instances are far away from the normal class's boundary because of some features which have an abnormal pattern (e.g. CURRENT_CONNECTIONS_4SEC) . For more information about these features, please take a look at the following features figures: Figure 4.26, 4.27, 4.30, 4.31, 4.32,4.34, 4.35, 4.36.

### *KDD Cup'99 Dataset's Features Extraction and Model Building*

Based on the results of selected features that are relevant with each label in [90] , we performed multiple experiments for each service, for each experiment the model was built on different combination of features, and applied on different tune values. In the following subsections we presented the feature selection of each service and the base line experiment results. We chose 10% training dataset, shown in Table 2.1, in the feature selection process.

#### Feature Selection and Model Building of KDD Cup'99 HTTP Service Dataset

The dataset used to perform the base line experiment is shown in Table 5.9. As shown in the experiments results in Table 5.15, the eight experiment appeared to have the highest accuracy rate 99.85% at the tune value 15. We chose the dataset features of experiment 8 to build OCC model for the KDD Cup'99 HTTP service.

The selected features, shown in Table 5.16, have achieved high accuracy rate as shown in Table 5.17. Where the detection accuracy of the normal instances reaches 97.73%.

**Table 5.15 KDD Cup'99 HTTP OCC feature selection experiments' accuracy results**

| The standard deviation of HTTP OCC model built on experiment 8 is 3.78. | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TUNE | Experiment Id | | | | | | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 90.34 | 89.85 | 89.11 | 89.09 | 90.39 | 90.33 | 91.76 | **94.26** | 90.97 | 91.08 | 91.04 | 91.02 | 91.09 | 91.46 | 92.39 |
| **1** | 96.17 | 94.39 | 92.58 | 92.56 | 93.30 | 93.73 | 94.25 | **95.36** | 93.67 | 93.69 | 93.67 | 93.66 | 93.54 | 94.43 | 94.74 |
| **1.5** | 97.27 | 97.14 | 96.85 | 96.84 | 97.26 | 97.27 | 97.77 | **95.59** | 97.37 | 97.40 | 97.55 | 97.56 | 97.41 | 98.69 | 99.43 |
| **2** | 97.90 | 97.86 | 97.66 | 97.65 | 97.96 | 97.97 | 98.24 | **96.16** | 98.06 | 98.09 | 98.11 | 98.09 | 97.98 | 99.41 | 98.77 |
| **3** | 98.87 | 98.86 | 98.71 | 98.71 | 98.92 | 98.92 | 98.98 | **96.46** | 98.83 | 98.84 | 98.85 | 98.86 | 98.75 | 98.59 | 98.55 |
| **4** | 98.53 | 98.70 | 98.71 | 98.71 | 98.84 | 98.85 | 99.02 | **96.56** | 99.17 | 99.19 | 99.20 | 99.22 | 99.33 | 98.35 | 98.28 |
| **5** | 98.06 | 98.20 | 98.32 | 98.32 | 98.41 | 98.41 | 98.56 | **97.78** | 98.61 | 98.62 | 98.62 | 98.62 | 98.77 | 98.08 | 98.03 |
| **6** | 97.90 | 97.97 | 97.98 | 97.97 | 98.04 | 98.04 | 98.11 | **98.94** | 98.17 | 98.18 | 98.18 | 98.18 | 98.26 | 98.05 | 98.04 |
| **7** | 97.88 | 97.97 | 97.98 | 97.98 | 98.04 | 98.04 | 98.05 | **99.49** | 98.05 | 98.06 | 98.06 | 98.06 | 98.06 | 98.04 | 98.03 |
| **8** | 97.89 | 97.97 | 97.97 | 97.97 | 98.02 | 98.02 | 98.03 | **99.70** | 98.03 | 98.04 | 98.04 | 98.04 | 98.04 | 98.04 | 98.03 |
| **9** | 97.91 | 97.97 | 97.99 | 97.99 | 98.03 | 98.03 | 98.03 | **99.73** | 98.03 | 98.04 | 98.04 | 98.04 | 98.04 | 98.04 | 98.03 |
| **10** | 97.97 | 98.01 | 97.99 | 97.99 | 98.03 | 98.03 | 98.03 | **99.80** | 98.03 | 98.04 | 98.04 | 98.04 | 98.04 | 97.98 | 97.96 |
| **11** | 97.90 | 97.99 | 97.93 | 97.93 | 98.03 | 98.03 | 98.03 | **99.81** | 98.05 | 97.98 | 97.98 | 97.98 | 98.00 | 97.93 | 98.01 |
| **12** | 97.89 | 97.93 | 97.90 | 97.90 | 97.99 | 97.99 | 97.99 | **99.82** | 97.99 | 97.93 | 97.93 | 97.93 | 97.93 | 97.89 | 97.95 |
| **13** | 97.86 | 97.92 | 97.86 | 97.86 | 97.95 | 97.95 | 97.95 | **99.84** | 97.95 | 97.89 | 97.89 | 97.89 | 97.91 | 97.87 | 97.93 |
| **14** | 97.83 | 97.88 | 97.82 | 97.82 | 97.91 | 97.91 | 97.93 | **99.84** | 97.93 | 97.87 | 97.87 | 97.87 | 97.87 | 97.83 | 97.89 |
| **15** | 97.84 | 97.86 | 97.80 | 97.80 | 97.89 | 97.89 | 97.89 | **99.85** | 97.89 | 97.83 | 97.83 | 97.83 | 97.83 | 97.83 | 97.89 |

**Table 5.16 KDD Cup'99 HTTP OCC selected feature set**

| # | Feature Name | # | Feature Name |
|---|---|---|---|
| 1 | Flag | 11 | srv_rerror_rate |
| 2 | logged_in | 12 | same_srv_rate |
| 3 | src_bytes | 13 | srv_diff_host_rate |
| 4 | Hot | 14 | diff_srv_rate |
| 5 | num_compromised | 15 | dst_host_count |
| 6 | count_v | 16 | dst_host_srv_count |
| 7 | serror_rate | 17 | dst_host_diff_srv_rate |
| 8 | srv_serror_rate | 18 | dst_host_same_src_port |

Based on the plot of Table 5.17, shown in Figure 5.4, the detection accuracy of normal class is increased whenever the tune value is increased and the detection accuracy of attacks is decreased.

As shown in Table 5.17 and Figure 5.4, the optimal tune value equals 6 at which we have 97.73% detection rate for normal class, and 100% detection rate for attack classes. At tune value 7 the detection rate of ipsweep, and portsweep attacks became decreasing. An R2L attack called phf. stay in 100% detection rate until the tune value 22, it decreased rapidly to 25%.

**Table 5.17 KDD Cup'99 base line experiment results of HTTP service**

| Label | Tune Value |
|---|---|

| | 0 | 1 | 3 | 5 | 6 | 7 | 10 | 15 | 21 | 22 | 23 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| normal. | 0.8389 | 0.8698 | 0.9004 | 0.9378 | **0.9773** | 0.9856 | 0.9944 | 0.9958 | 0.9965 | 0.9969 | 0.9970 | 0.9971 |
| back. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| ipsweep. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 0.9787 | 0.9681 | 0.9681 | 0.9681 | 0.9681 | 0.9681 | 0.9681 |
| neptune. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| phf. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.2500 | 0.0000 | 0.0000 |
| portsweep. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 0.9990 | 0.9971 | 0.9933 | 0.9875 | 0.9875 | 0.9875 | 0.9827 |
| satan. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |

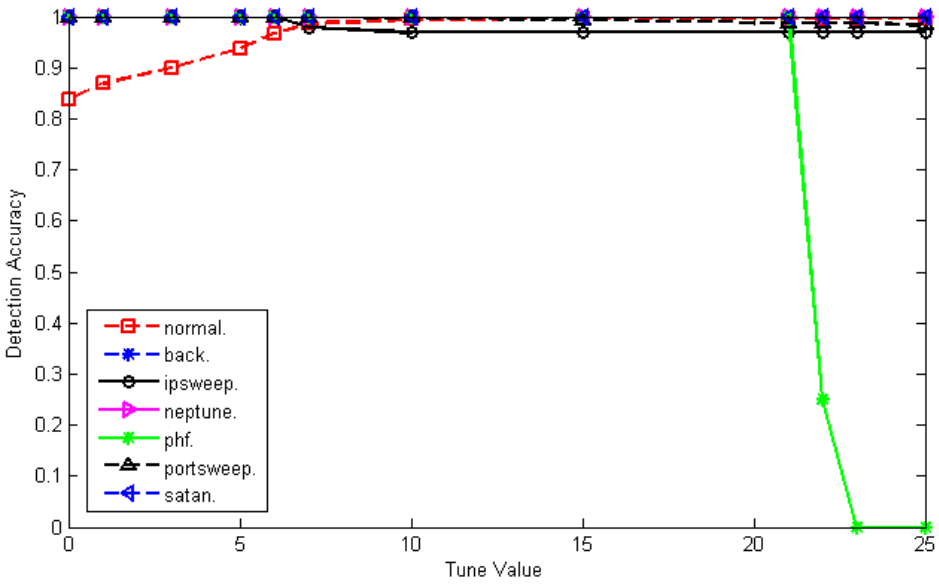


**Figure 5.4 KDD Cup'99 HTTP service's base line experiment result chart**

The optimal tune value based on ROC curve found at a false positive rate equals 0% as shown in Figure 5.5. At that point the true positive rate was 97.73% which is the best detection rate compared with the false positive rate. The increase of false positive rate was due to the decrease of both Probe attacks portsweep and ipsweep.

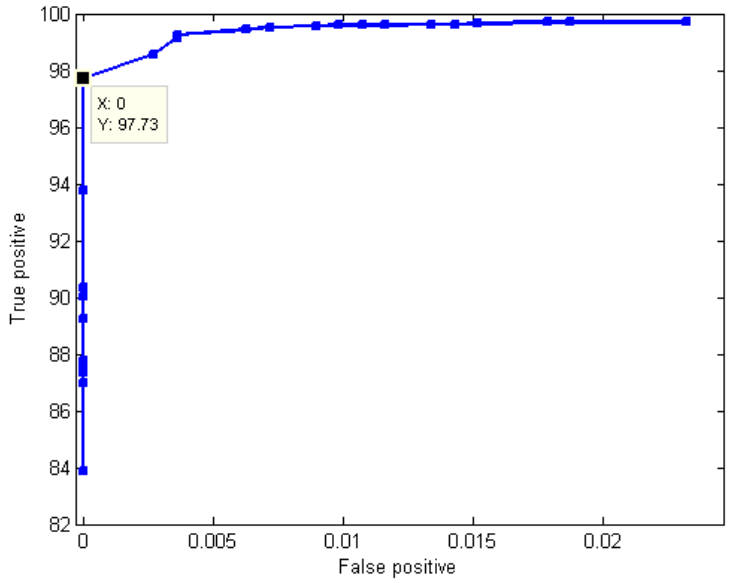


**Figure 5.5 KDD Cup'99 HTTP service's ROC chart of base line experiment**

### Feature Selection and Model Building of ECR_I Service

The dataset used to perform the base line experiment is shown in Table 5.10. As shown in the experiments results in Table 5.18, the nine experiment appeared to have the highest accuracy rate 99.99% at the tune value 15. We chose the dataset features of experiment 9 to build OCC model for the KDD Cup'99 ECR_I service.

**Table 5.18 KDD Cup'99 ECR_I OCC feature selection experiments' accuracy results**

| The standard deviation of ECR_I OCC model built on experiment 9 is 1.414. | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **TUNE** | **Experiment Id** | | | | | | | | |
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **0** | 87.246 | 92.029 | 93.768 | 92.029 | 92.029 | 87.246 | 87.246 | 87.246 | **93.623** |
| **1** | 95.362 | 95.507 | 95.507 | 95.507 | 95.507 | 95.794 | 95.794 | 95.362 | **96.957** |
| **2** | 95.942 | 95.942 | 95.942 | 95.942 | 95.942 | 96.374 | 96.374 | 95.942 | **97.681** |
| **3** | 96.087 | 96.087 | 96.087 | 96.087 | 96.087 | 96.374 | 96.374 | 96.232 | **98.261** |
| **4** | 96.231 | 96.231 | 96.231 | 96.231 | 96.231 | 96.808 | 96.808 | 97.681 | **99.130** |
| **5** | 97.536 | 97.536 | 97.681 | 97.536 | 97.536 | 97.967 | 97.967 | 98.550 | **99.130** |
| **6** | 98.260 | 98.260 | 98.260 | 98.260 | 98.260 | 98.690 | 98.690 | 98.840 | **99.130** |
| **7** | 98.550 | 98.695 | 98.695 | 98.695 | 98.695 | 99.124 | 99.124 | 99.130 | **99.130** |
| **8** | 98.985 | 98.984 | 98.984 | 98.984 | 98.984 | 99.552 | 99.552 | 99.130 | **99.130** |
| **11** | 98.984 | 98.984 | 98.984 | 98.984 | 98.984 | 99.545 | 99.545 | 99.129 | **99.129** |
| **12** | 98.984 | 98.984 | 98.984 | 98.984 | 98.984 | 99.544 | 99.544 | 99.129 | **99.129** |
| **13** | 98.984 | 98.984 | 98.984 | 98.984 | 98.984 | 99.689 | 99.689 | 99.129 | **99.129** |
| **14** | 99.564 | 99.564 | 99.564 | 99.564 | 99.564 | 99.675 | 99.675 | 99.419 | **99.419** |
| **15** | 99.564 | 99.564 | 99.564 | 99.564 | 99.564 | 99.675 | 99.675 | 99.709 | **99.999** |

The selected features, shown in Table 5.19, have achieved high accuracy rate as shown in Table 5.20. Where the detection accuracy of the normal instances reaches 98.26%.

**Table 5.19 KDD Cup'99 ECR_I OCC selected feature set**

| # | Feature Name | # | Feature Name |
|---|---|---|---|
| 1 | src_bytes | 4 | serror_rate |
| 2 | dst_bytes | 5 | diff_srv_rate |
| 3 | wrong_fragment | 6 | dst_host_count |

Based on the plot of Table 5.20, shown in Figure 5.6, the detection accuracy of normal class is increased whenever the tune value is increased and the detection accuracy of attacks is decreased. As shown in Figure 5.6, the DoS attacks, smurf and pod, never decreased even if the normal instances reached 100% detection rate.

As shown in Table 5.20 and Figure 5.6, the optimal tune value equals 4 at which we have 98.26% detection rate for normal class, and 100% detection rate for attack classes At tune value 5 the detection rate of the probe attacks, ipsweep dropped to70.9% , followed by portsweep, and nmap which have a detection rate of 0.0% at tune value 9, while the detection DoS attacks remained approximately 100%.

**Table 5.20 KDD Cup'99 base line experiment results of ECR_I service**

| Label | Tune Value | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **8** | **9** | **13** | **14** | **15** |
| normal. | 0.8725 | 0.9391 | 0.9536 | 0.9652 | **0.9826** | 0.9826 | 0.9826 | 0.9826 | 0.9826 | 0.9884 | 1.0000 |
| ipsweep. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 0.7095 | 0.7095 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| nmap. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| pod. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| portsweep. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| smurf. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |

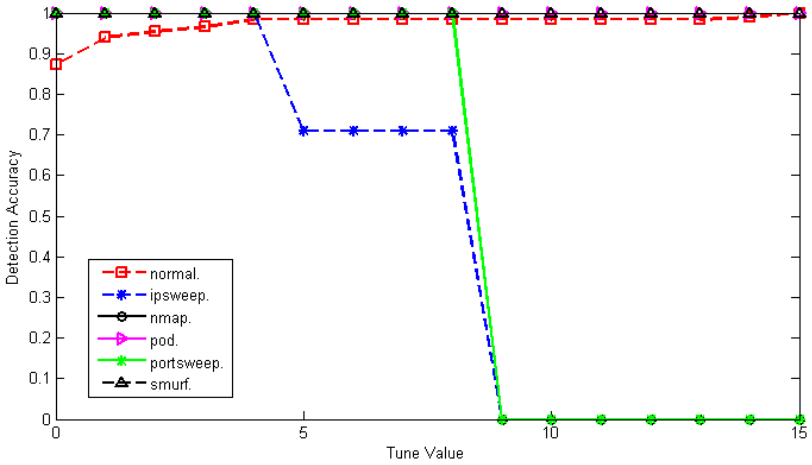


**Figure 5.6 KDD Cup'99 ECR_I service's base line experiment result chart**

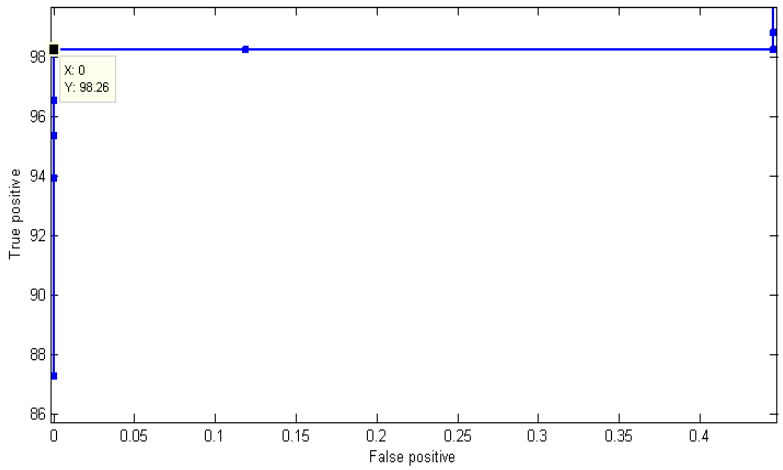


**Figure 5.7 KDD Cup'99 ECR_I service's ROC chart of base line experiment**

The optimal tune value based on ROC curve found at a false positive rate equals 0% as shown in Figure 5.7. At that point the true positive rate was 98.26% which is the best detection rate compared with the false positive rate. As shown in Figure 5.7, the false positive rate increasing without any increasing in true positive rate. This means that 1.74% of normal instances are far away from the normal class boundary.

### Feature Selection and Model Building of POP3 Service

The dataset used to perform the base line experiment is shown in Table 5.11. As shown in the experiments results in Table 5.21, experiment 13 appeared to have the highest accuracy rate 98.78% at the tune value 5. Although experiments 12 and 14 achieved a detection rate of 87.78, we discard them because experiment 13 achieved a higher accuracy rate at a minimum increase in tune value which equals 1. We chose the dataset features of experiment 13 to build OCC model for the KDD Cup'99 POP3 service.

The selected features, shown in Table 5.22, achieved high accuracy rate as shown in Table 5.23. Where the detection accuracy of the normal instances reaches 97.21% at tune value 1.

**Table 5.21 KDD Cup'99 POP3 OCC feature selection experiments' accuracy results**

| **The standard deviation of POP3 OCC model built on experiment 13 is 2.236.** | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TUNE** | **Experiment Id** | | | | | | | | | | | | | |
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| **0** | 86.08 | 86.08 | 86.08 | 86.08 | 90.51 | 93.04 | 89.24 | 87.34 | 89.24 | 88.61 | 89.87 | 90.51 | **90.51** | 92.41 |
| **1** | 92.41 | 92.41 | 92.41 | 91.77 | 94.94 | 94.76 | 92.86 | 92.41 | 93.67 | 88.79 | 95.57 | 95.57 | **97.21** | 95.62 |
| **1.5** | 93.04 | 93.04 | 93.04 | 93.04 | 96.43 | 96.02 | 94.12 | 93.90 | 96.84 | 89.88 | 97.47 | 96.84 | **97.51** | 96.25 |
| **2** | 93.67 | 93.67 | 93.67 | 93.67 | 96.43 | 96.02 | 94.12 | 95.16 | 97.06 | 92.41 | 97.69 | 97.06 | **97.51** | 96.25 |
| **2.5** | 93.67 | 93.67 | 93.67 | 93.67 | 96.43 | 96.02 | 94.12 | 95.16 | 97.06 | 92.23 | 98.33 | **98.33** | **97.51** | 96.25 |
| **3** | 93.67 | 93.67 | 93.67 | 93.67 | 97.06 | 96.02 | 95.39 | 95.16 | 97.06 | 91.42 | 98.55 | **97.92** | **97.51** | 96.25 |
| **3.5** | 93.67 | 93.67 | 93.67 | 93.67 | 97.06 | 96.66 | 95.39 | 95.16 | 97.06 | 90.20 | 98.55 | 97.92 | **97.51** | 96.25 |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **4** | 95.57 | 95.57 | 95.57 | 95.57 | 97.06 | 96.66 | 95.39 | 96.43 | 97.06 | 90.83 | 98.55 | 97.92 | **97.51** | **97.51** |
| **4.5** | 96.84 | 96.84 | 96.84 | 96.20 | 97.92 | 97.29 | 97.29 | 97.29 | 97.92 | 90.83 | 98.55 | 98.55 | **97.51** | **97.23** |
| **5** | 97.47 | 97.47 | 97.47 | 97.47 | 97.92 | 98.55 | 97.29 | 98.55 | 97.92 | 90.65 | 97.97 | 98.78 | **98.78** | 98.15 |
| **5.5** | 97.47 | 97.47 | 97.47 | 97.47 | 98.55 | 98.15 | 96.88 | 98.55 | 98.55 | 90.65 | 97.97 | 98.78 | **98.78** | 98.78 |
| **6** | 98.10 | 98.10 | 98.10 | 98.10 | 98.55 | 98.15 | 96.88 | 98.55 | 98.55 | 90.65 | 97.97 | 98.78 | **98.78** | 98.78 |
| **6.5** | 98.73 | 98.73 | 98.73 | 98.73 | 98.15 | 98.15 | 96.88 | 98.15 | 98.15 | 90.65 | 97.97 | 98.78 | **98.78** | 98.78 |
| **7** | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 97.51 | 98.71 | 98.73 | 90.24 | 97.97 | 98.78 | **98.78** | 98.78 |
| **7.5** | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 97.51 | 98.71 | 98.73 | 90.24 | 97.97 | 98.78 | **98.78** | 98.78 |
| **8** | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 97.51 | 98.71 | 98.73 | 90.24 | 97.97 | 98.78 | **98.78** | 98.78 |
| **8.5** | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 97.51 | 98.71 | 98.73 | 90.24 | 97.97 | 98.78 | **98.78** | 98.78 |
| **9** | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 97.51 | 98.72 | 98.73 | 90.24 | 97.97 | 98.78 | **98.78** | 98.78 |
| **9.5** | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 97.51 | 98.72 | 98.73 | 90.24 | 97.97 | 98.78 | **98.78** | 98.78 |
| **10** | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 98.73 | 97.51 | 98.72 | 98.73 | 90.24 | 97.97 | 98.78 | **98.78** | 98.78 |

**Table 5.22 KDD Cup'99 POP3 OCC selected feature set**

| # | Feature Name | # | Feature Name |
|---|---|---|---|
| 1 | Flag | 7 | srv_rerror_rate |
| 2 | num_failed_logins | 8 | srv_diff_host_rate |
| 3 | count_v | 9 | dst_host_count |
| 4 | serror_rate | 10 | dst_host_srv_count |
| 5 | srv_serror_rate | 11 | dst_host_srv_diff_host_rate |
| 6 | rerror_rate | 12 | dst_host_srv_serror_rate |

Based on the plot of Table 5.23, shown in Figure 5.8, the detection accuracy of normal class is increased whenever the tune value is increased and the detection accuracy of attacks is decreased. As shown in Figure 5.8, the neptune attack, which is a DoS attack, never decreased and the normal detection rate increased until reached 100% detection rate. While a probe attack called portsweep decrease rapidly from 99.5% then 34.94% then 10.11 % at tune values 0,1 and 1.5 respectively.

**Table 5.23 KDD Cup'99 base line experiment results of POP3 service**

| Label | Tune Value | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **1.5** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| normal. | 0.8101 | **0.9721** | 0.9747 | 0.9747 | 0.9747 | 0.9747 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| neptune. | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| nmap. | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| portsweep. | 0.9952 | **0.3494** | 0.1011 | 0.1011 | 0.1011 | 0.1011 | 0.1011 | 0.1011 | 0.1011 | 0.1001 | 0.1001 |
| satan. | 1.0000 | **0.9993** | 0.9993 | 0.9993 | 0.9979 | 0.9979 | 0.9965 | 0.9965 | 0.9965 | 0.9965 | 0.9965 |

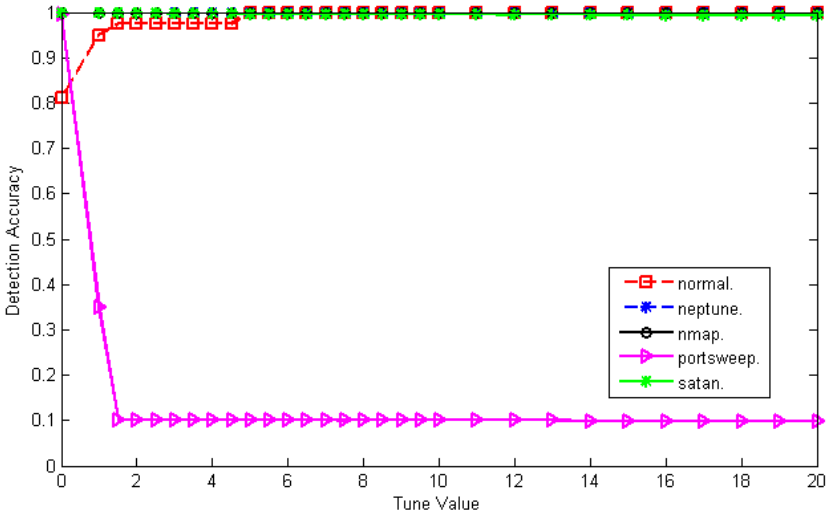


**Figure 5.8 KDD Cup'99 POP3 service's base line experiment result chart**

As shown in Table 5.23 and Figure 5.8, the optimal tune value equals 1 at which we have 97.2 detection rate for normal class, and 100% detection rate for attack classes except for portsweep attack which is a probe attack that achieved 34.94% detection rate. At tune value 1.5 the detection rate of the probe attack portsweep became 10.22%, while the detection DoS attack, neptune, remained approximately 100%.

The optimal tune value based on ROC curve found at a false positive rate equals 0.617% as shown in Figure 5.9. At that point the true positive rate was 97.2% which is the best detection rate compared with the false positive rate. The next false positive rate found on ROC was 0.85% which is a high rate raised because of a drop in detection rate of portsweep probe attack which reached 10.11% which decreased at tune value equals 1.5.

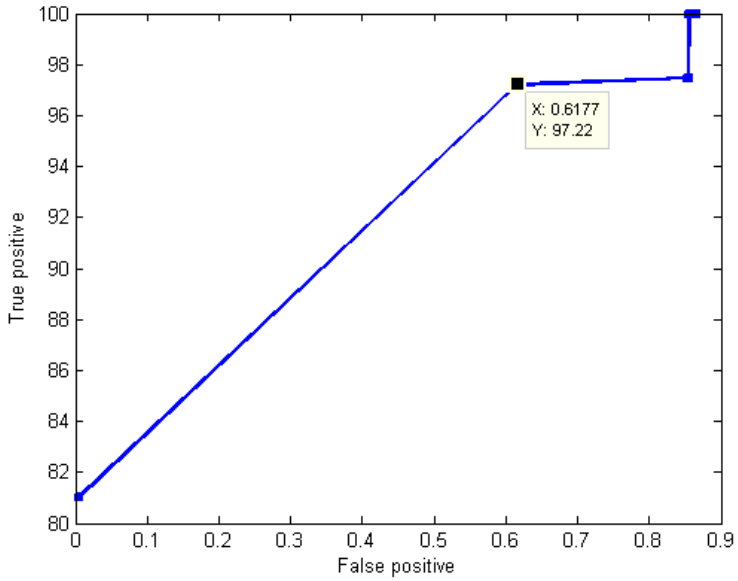


**Figure 5.9 KDD Cup'99 POP3 service's ROC chart of base line experiment**

## *5.5Proof of Concept Evaluation of the Model*

The effectiveness of the model is evaluated by its capability to make accurate predictions. According to the real nature of a given event compared to the prediction from the model, four possible outcomes are shown in Table 5.24, known as the Confusion Matrix [92].

Confusion Matrix is used to evaluate the model. Columns and rows of the matrix represent actual label and the instance of predicate label, respectively.

The following four parameters define the member of matrix:

- True positive (TP) is shown in (Eq. 5.3).
- True negative (TN) is shown in (Eq. 5.4).
- False positive (FP) is shown in (Eq. 5.5).
- False negative (FN) is shown in (Eq. 5.6).

The accuracy of the model is considered to be the most commonly measurement used to evaluate the performance. Accuracy (Eq. 5.8), detection rate (Eq. 5.7), false positive rate (Eq.5.5) and false alarm rate (Eq. 5.6) were used in this research for model evaluation.
Confusion Matrix: is created after classification process. The main elements in the matrix are shown in Table 5.24.

**True positive (TP)** refers to positive instances that correctly labeled by the classifier (When normal data detected as normal).

True Positive rate = TP / ( TP + FN ) …… (Eq. 5.3).

**Table 5.24 Confusion Matrix Structure**

| | | **Actual (True) Class** | |
|---|---|---|---|
| | | **Actual Normal (Positive)** | **Actual Abnormal (Negative)** |
| **Predicate Class** | **Predicate Normal (Positive)** | True positive (TP) | False positive (FP) |
| | **Predicate Abnormal (Negative)** | False negative (FN) | True negative (TN) |

**True negative (TN)** refers to negative instances that correctly labeled the classifier (when abnormal data detected as abnormal).

True Negative rate = TN / ( TN + FP ) ……..(Eq. 5.4).

**False Positive (FP)** is the negative instances that were incorrectly labeled (when abnormal data detected as normal)

False Positive rate = FP / ( FP + TN ) …….(Eq. 5.5).

**False Negative (FN)/ False Alarm Rate (FAR)** is the positive instances that were incorrectly labeled (when normal data detected as abnormal)

False Positive rate = FN / ( FN + TP ) ……..(Eq. 5.6).

**Detection Rate (DR)** is the percentage of positive instances that correctly labeled by the classifier (i.e. the proportion of true positives which are correctly identified and the proportion of true negative which are correctly identified). Assume that N: number of normal, A:number of abnormal.

Detection Rate = (TP*N + TN*A) / (N + A) ……..... (Eq. 5.7).

**Accuracy** is the percentage of test set tuples that are correctly classified by classifier (i.e. the proportion of true results in the population).

Accuracy = (TP + TN) / (TP + TN + FP +FN) ……..... (Eq. 5.8).

**F-measure:** refer to the harmonic mean of precision and recall

F-Measure = 2*TP/(2*TP+FP+FN) ---------- (Eq. 5.9)

**Correlation:** is a measure of how predictions correlate with actual data. This ranges from -1 to 1 where a correlation coefficient of 1 corresponds to predictions that perfectly match class labels, and a coefficient of 0 corresponds to random guessing.

$$\text{Correlation} = \frac{TP.TN - FP.FN}{\sqrt{(TP+FN).(TP+FP).(TN+FP).(TN+FN)}}$$ ---------- (Eq. 5.10)

An applied example of how to calculate these measurements, suppose we have the following Table 5.25 filled from the classification results.

**Table 5.25 Confusion Matrix Example**

| | | **Actual (True) Class** | |
|---|---|---|---|
| | | **Actual Normal (Positive)** | **Actual Abnormal (Negative)** |
| **Predicate Class** | **Predicate Normal (Positive)** | 6954 | 46 |
| | **Predicate Abnormal (Negative)** | 412 | 588 |

True Positive rate = TP/(TP+FN)= 6954/(6954+412) =0.94

True Negative rate = TN/(TN+FP) = 588/(588+46) =0.927

Accuracy = (TP + TN) / (TP + TN + FP +FN) = (6854 + 588)/7542= 0.9867

## *5.6 summary*

In this chapter, the proposed model methodology was presented and explained. We also presented the methodology steps to achieve our primary model. Then we gave a description of the collected datasets (BM-AUN2015 and KDD Cup'99), and a description of their features. After that, the preprocessing steps and feature selection of each dataset were described. Feature selection experiments were performed and listed their results and chose the optimal features set of each service, then the standard deviation of each service is built based on the selected features. Baseline experiments were performed to obtain the optimal tune value by which we need to validate the model in the testing phase in the next chapter. The measurements needed to evaluate the accuracy of our model was presented.

# Chapter 6: Experimental Results Discussion and Evaluation

In this chapter, the experiments results of both dataset, KDD Cup'99 and BM-AUN2015, were presented and analyzed. The tools, requirements and environments used in our model were explained. Main evaluation measurements such as accuracy, detection rate and false alarm were calculated after each experiment and an overall measurements of these metric were also performed.

## *6.1 Experiments Setup:*

This section describes the experiments environment and tools used to measure and evaluate the performance of the proposed model.

### 6.1.1 Experimental Environments and Tools:

The experiments were conducted using an Intel® Core™2 Duo CPU 1.8GHz with 2.5GB RAM. Special programs were used for constructing the model and implementation of model functions, such as

**RapidMiner Studio Program:** RapidMiner [93] is an international open-source data mining framework. It enables users to model complex knowledge discovery processes as it supports nested operator chains. Graphical User Interface feature of RapidMiner enables it to be used for complex process modeling. Moreover, it can be used as a library in other programs.

RapidMiner is commonly used as a data mining tool for many reasons. First, it has many data loading, modeling, preprocessing and visualization methods that avoid the trouble of preprocessing data sets and help to visualize the results. It is easy to use the currently robust graphical user interface that facilitates the modeling of different complex processes. Second, it is modular and thus allows using some functionalities for the extension, for example, using distance measurements for anomaly detection operators. Finally, it is easily extensible and was used for clustering data and construct network traffic behavior using K-Means algorithm and Decision Tree, respectively.

**Oracle Database10g (SQL+PL/SQL):** Oracle Software [94] was used for normal service behavior's standard deviation calculation and labeling process. The main evaluations measurements equations were implemented. Testing PL/SQL code is listed in Appendix A.

### 6.1.2 Experimental Measurements:

The evaluation measures of our model performance are based on Confusion Matrix. Accuracy rate, detection rate and false alarm were used. These measures have been described in section 5.5.

## *6.2 BM-AUN2015 Experiments Cases and Results:*

Four experiments cases were conducted. Three of these experiments were performed on three different application layer attacks and the fourth experiment were performed on network layer attacks in addition to the normal dataset ,Day1, Day2, Day4, as shown in Table 4.5. Day3 were used as the normal service behavior which was used to build the service centroid table. The service centroid table was built and the standard deviation of it was calculated as described in base line experiment in section 5.4.1. The process of preparing our model is shown in Figure 5.3 (a).

### 6.2.1 Experiment Case 1, SlowRead Attack:

This experiment is performed on SlowRead attack, which is one of the application layer attacks. This attack was performed in different scenarios as shown in section 4.2.1.2. we used Day3 normal dataset as the model centroid table and Day1,Day2 and Day4 were used for testing as shown in Table 6.1.

**Table 6.1 BM-AUN2015 experiment case1, SlowRead attack dataset**

| | Normal | | | | Attack | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Application layer attack | | | |
| | Day3 Centroid | Day1 | Day2 | Day4 | SlowRead Scenario1 | SlowRead Scenario2 | SlowRead Scenario3 | SlowRead Scenario4 |
| **# instances** | **13,548** | 703 | 3,296 | 10,332 | 2,960 | 2,995 | 214 | 116 |
| ∑ | **13,548** | **14,331** | | | **6,285** | | | |

**Experiment Results**

The results of this experiment, shown in Table 6.2, shows that the detection rate of attack instances is 100% with 0.0% false positive and 2.39 % false negative at tune value 5.5. The optimal tune value of this experiment is at tune value equals 7 where the true positive rate reaches 99.913% with false negative rate 0.086% where as the true negative rate reaches 99.966% with false positive rate 0.033% as shown in Table 6.3.

Figure 6.1 shows an extreme drop in detection rate at tune value equals 10 for SlowRead scenario3 and scenario4 attacks, which are a type of DDoS attacks, and decrease rapidly until it reaches 0.0% true negative rate . The drop in attack detection rate begins from the tune value 9. Table 6.2 BM-AUN2015 experiment case1, SlowRead attack results

| | Detection Rate | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Label/Tune Value** | **0** | **2** | **4** | **6** | **7** | **8** | **9** | **10** | **14** | **16** | **30** |
| Normal | 0.843 | 0.959 | 0.977 | 0.978 | **0.999** | 0.999 | 0.999 | 0.999 | 0.999 | 1.000 | 1.000 |
| SlowRead-Scenario1 | 1.000 | 1.000 | 1.000 | 1.000 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| SlowRead-Scenario2 | 1.000 | 1.000 | 1.000 | 1.000 | **0.999** | 0.997 | 0.997 | 0.994 | 0.985 | 0.980 | 0.951 |
| SlowRead-Scenario3 | 1.000 | 1.000 | 1.000 | 1.000 | **1.000** | 1.000 | 0.897 | 0.780 | 0.449 | 0.042 | 0.000 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SlowRead-Scenario4 | 1.000 | 1.000 | 1.000 | 1.000 | **1.000** | 0.991 | 0.826 | 0.730 | 0.435 | 0.026 | 0.000 |

As shown in Figure 6.1, the drop of attack detection rate begins from a tune value of 9. This means that there's an instance similarity between normal and attack instances at this expanding value of normal class boundary and greater.

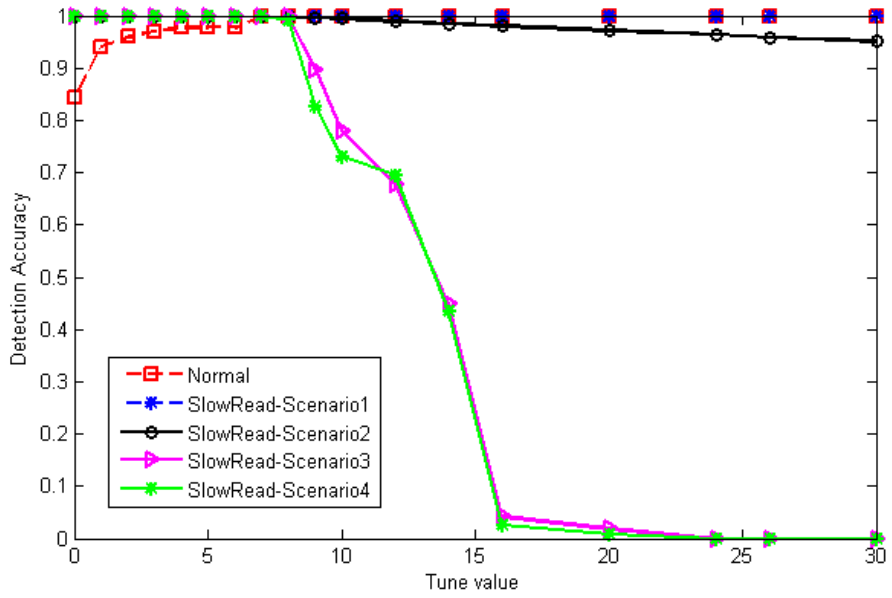


**Figure 6.1 BM-AUN2015 experiment case 1, SlowRead attack results chart**

## Experiment Evaluation

Table 6.3 shows the confusion matrix results at the tune value 7 which is the optimal standard deviation expanding value with high correlation rate. Based on Table 6.3, the model has achieved 99.92% detection rate, 99.94% accuracy rate and a false positive rate 0.033% with a correlation rate reaches 99.9% and F-Measure equals to 99.9%.

**Table 6.3 BM-AUN2015 experiment case1, SlowRead attack confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 84.328 | 0.000 | 100.000 | 15.672 | 86.866 | 7.836 | 92.164 | 0.854 | 0.915 |
| 2 | 95.927 | 0.000 | 100.000 | 4.073 | 96.586 | 2.037 | 97.963 | 0.960 | 0.979 |
| 3 | 97.053 | 0.000 | 100.000 | 2.947 | 97.531 | 1.473 | 98.527 | 0.971 | 0.985 |
| 4 | 97.718 | 0.000 | 100.000 | 2.282 | 98.087 | 1.141 | 98.859 | 0.977 | 0.988 |
| 5.5 | 97.761 | 0.000 | 100.000 | 2.239 | 98.124 | 1.119 | 98.881 | 0.978 | 0.989 |
| 6 | 97.761 | 0.017 | 99.983 | 2.239 | 98.119 | 1.128 | 98.872 | 0.978 | 0.989 |
| **7** | **99.913** | **0.033** | **99.967** | **0.087** | **99.917** | **0.060** | **99.940** | **0.999** | **0.999** |
| 8 | 99.913 | 0.284 | 99.716 | 0.087 | 99.904 | 0.185 | 99.815 | 0.996 | 0.998 |
| 9 | 99.928 | 7.001 | 92.999 | 0.072 | 99.766 | 3.537 | 96.463 | 0.934 | 0.967 |
| 10 | 99.928 | 12.372 | 87.628 | 0.072 | 99.622 | 6.222 | 93.778 | 0.887 | 0.945 |
| 14 | 99.942 | 28.299 | 71.701 | 0.058 | 99.191 | 14.179 | 85.821 | 0.762 | 0.889 |
| 16 | 99.957 | 48.797 | 51.203 | 0.043 | 98.697 | 24.420 | 75.580 | 0.559 | 0.834 |
| 30 | 99.986 | 51.219 | 48.781 | 0.014 | 98.462 | 25.617 | 74.383 | 0.484 | 0.827 |

### Experiment Results Discussion

The number of normal instances that fall out of the normal class boundary at tune value equals 7 are 8 instances listed in Table 6.4. These instances have a distance greater than the expanded standard deviation which is 10.698. As shown in Table 6.4, the shaded bold cells have extreme values based on BM-AUN2015 dataset analysis as shown in section 4.5. Note that SlowRead-scenario3 and SlowRead-scenario4 attacks, based on the standard deviation of normal class, have a minimum distance from normal class= 11.42 up to = 23.68.

**Table 6.4 BM-AUN2015 experiment case1, extreme normal instances in SlowRead dataset**

| The standard deviation = 2.698 + 8 = 10.698 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| FLOWKEY | NUMBER OF SERVER ACK | IS HTTP SESSION | IS HTTP HEADER END | NUMBER OF CLIENT TCP PSH | AVG TCP PAYLOAD LENGTH | AVG CURRENT CONNECTIONS 4SEC | NUMBER ZERO WINDOW PKTS | DISTANCE |
| 76926 | 1 | 1 | 1 | 2 | 673 | 2 | **77** | 112.98 |
| 80015 | **3** | **0** | **0** | 0 | **0** | 1 | 0 | 45.63 |
| 75357 | **3** | **0** | **0** | 0 | **0** | 4 | 0 | 45.48 |
| 83669 | **2** | 1 | 1 | 2 | 400 | 12 | 0 | 22.69 |
| 82931 | **2** | 1 | 1 | 3 | 397 | 11 | 0 | 22.68 |
| 84633 | 1 | 1 | 1 | **112** | 1395.105 | 3 | 0 | 17.97 |
| 84691 | 1 | 1 | 1 | **87** | 1297.605 | 5 | 0 | 14.14 |
| 83099 | 1 | 1 | 1 | **73** | 1394.762 | 2 | 0 | 12.31 |

As shown in Table 6.4, Flowkeys 80015, 75357, 83669 and 82931 didn't complete the three way handshaking, they seem like SYNC attack but generally they are normal to happen because of network congestion as an example. On the other hand Flowkey 76926 looks like it has a congestion problem because of the number of Zero TCP WINDOW SIZE packets. The last three instances, 84633, 84691 and 83099, are normal instances, but because of the extreme number of client TCP-PSH flag compared with the average TCP payload length their distance is raised from the normal class.

## 6.2.2 Experiment Case 2, SlowPost Attack:

This experiment was performed on SlowPost attack, which is one of the application layer attacks. This attack was performed in different scenarios as shown in subsection 4.2.1.2.

**Table 6.5 BM-AUN2015 experiment case2, SlowPost attack dataset**

| | Normal | | | | Attack | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Application layer attack | | | |
| | Day3 Centroid | Day1 | Day2 | Day4 | SlowPost Scenario1 | SlowPost Scenario2 | SlowPost Scenario3 | SlowPost Scenario4 |
| **#instances** | **13,548** | 703 | 3,296 | 10,332 | 2,977 | 2,983 | 262 | 193 |

| | | | |
|---|---|---|---|
| ∑ | 13,548 | 14,331 | 6,415 |

We used Day3 normal dataset as the model centroid's table and Day1,Day2 and Day4 were used for testing as shown in Table 6.5.

### Experiment Results

The results of this experiment, shown in Table 6.6, shows that the detection rate of attack instances is 100% with 0.0% false positive and 3.58 % false negative. The optimal tune value of this experiment is at tune value equals 3.3 where the true positive rate reaches 97.62% with false negative rate 2.38% where as the true negative rate reaches 99.52% with false positive rate 0.484% as shown in Table 6.7.

**Table 6.6 BM-AUN2015 experiment case2, SlowPost attack results**

| | Detection Rate | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Label/Tune Value** | **0** | **2** | **3** | **3.3** | **6** | **8** | **10** | **12** | **16** | **24** | **30** |
| Normal | 0.843 | 0.959 | 0.971 | **0.976** | 0.978 | 0.999 | 0.999 | 0.999 | 1.000 | 1.000 | 1.000 |
| SlowPost-Scenario1 | 1.000 | 1.000 | 1.000 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| SlowPost -Scenario2 | 1.000 | 1.000 | 1.000 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 0.989 | 0.969 | 0.956 |
| SlowPost -Scenario3 | 1.000 | 1.000 | 1.000 | **0.996** | 0.885 | 0.718 | 0.347 | 0.046 | 0.000 | 0.000 | 0.000 |
| SlowPost -Scenario4 | 1.000 | 1.000 | 0.995 | **0.984** | 0.870 | 0.710 | 0.347 | 0.052 | 0.000 | 0.000 | 0.000 |

Figure 6.2 shows an extreme drop in detection rate at tune value equals 5 for SlowPost scenario3 and scenario4 attacks and decrease rapidly until it reaches 0.0% true negative rate. These attacks scenarios category is classified under DDoS attacks categories.

As shown in Figure 6.2, the drop of attack detection rate begins from a tune value of 5. This means that there's an instance similarity between normal and attack instances at the tune value 5 and greater.

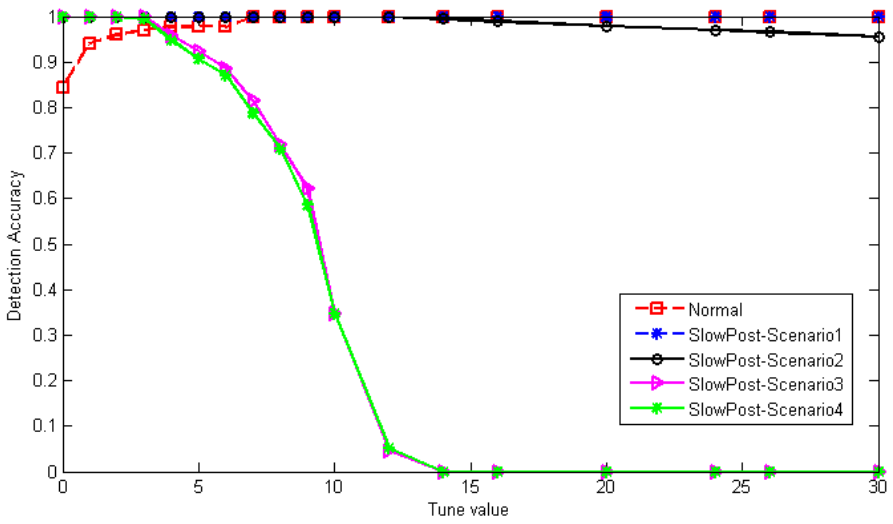


**Figure 6.2 BM-AUN2015 experiment case 2, SlowPost attack results chart**

### Experiment Evaluation

Table 6.7 shows the confusion matrix results at the tune value 3.3 which is the optimal standard deviation expanding value with high correlation rate. Based on Table 6.7, the model has achieved 97.999% detection rate, 98.566% accuracy rate and a false positive rate 0.484% with a correlation rate reaches 97.2% and F-Measure equals to 98.6%.

**Table 6.7 BM-AUN2015 experiment case2, SlowPost attack confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 84.328 | 0.000 | 100.000 | 15.672 | 86.935 | 7.836 | 92.164 | 0.854 | 0.915 |
| 2 | 95.927 | 0.000 | 100.000 | 4.073 | 96.604 | 2.037 | 97.963 | 0.960 | 0.979 |
| 2.5 | 96.418 | 0.000 | 100.000 | 3.582 | 97.014 | 1.791 | 98.209 | 0.965 | 0.982 |
| 2.6 | 96.447 | 0.130 | 99.870 | 3.553 | 97.034 | 1.841 | 98.159 | 0.964 | 0.981 |
| 3 | 97.053 | 0.130 | 99.870 | 2.947 | 97.540 | 1.538 | 98.462 | 0.970 | 0.984 |
| 3.1 | 97.082 | 0.259 | 99.741 | 2.918 | 97.561 | 1.588 | 98.412 | 0.969 | 0.984 |
| **3.3** | **97.617** | **0.484** | **99.516** | **2.383** | **97.999** | **1.434** | **98.566** | **0.972** | **0.986** |
| 4 | 97.718 | 2.345 | 97.655 | 2.282 | 98.024 | 2.314 | 97.686 | 0.954 | 0.977 |
| 6.5 | 99.740 | 7.355 | 92.645 | 0.260 | 99.553 | 3.808 | 96.192 | 0.928 | 0.964 |
| 7 | 99.913 | 9.986 | 90.014 | 0.087 | 99.613 | 5.037 | 94.963 | 0.907 | 0.954 |
| 12 | 99.942 | 47.568 | 52.432 | 0.058 | 98.437 | 23.813 | 76.187 | 0.577 | 0.839 |

### Experiment Results Discussion

The number of normal instances that fall out of the normal class boundary at tune value 5 are 164 instances included instances listed in Table 6.4. We have listed a representation of these instances in Table 6.8. These instances have a distance greater than the expanded standard deviation which is 6.698. As shown in Table 6.8, the shaded bold cells have extreme values based on BM-AUN2015 dataset analysis as shown in section 4.5. Note that SlowPost-scenario3 and SlowPost-scenario4 attacks, based on the standard deviation of normal class, have a minimum distance from normal class= 5.94 up to = 16.75.

** We have noted that the Flowkey 83004 shown in Table 6.8 has an extreme number of concurrent connections reaches 107 connections in a window of 2 second these connections after deep investigations came from a mobile user agent called "*Nokia306/2.0 (03.63) Profile/MIDP-2.1 Configuration/CLDC-1.1 UCWEB/2.0 (Java; U; MIDP-2.0; ar-SA; Nokia306) U2/1.0.0 UCBrowser/9.5.0.449 U2/1.0.0 Mobile*". This user agent open parallel connections to request the web page files to speed up its browsing. The number of instances of this case was 155 instance, we have listed one of them in Table 6.8 under Flowkey 83304. The relative extreme values of the other instances was number of TCP-PSH flag and TCP WINDOW SIZE. Two instances have an unusual increase in number of zero TCP WINDOW SIZE, it was may be due to network congestion.

**Table 6.8 BM-AUN2015 experiment case2, extreme normal instances in SlowPost dataset**

| The standard deviation = 2.698 + 4 = 6.698 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| FLOWKEY | NUMBER OF SERVER ACK | IS HTTP SESSION | IS HTTP HEADER END | NUMBER OF CLIENT TCP PSH | AVG TCP PAYLOAD LENGTH | AVG CLNT TCP WINDOW SIZE | AVG CURRENT CONNECTIONS 4SEC | NUMBER ZERO WINDOW PKTS | DISTANCE |
| 79832 | 1 | 1 | 1 | **70** | 408.04 | **916.74** | 3 | 0 | 10.39 |
| 82333 | 1 | 1 | 1 | 3 | 444.33 | **4589.88** | 3 | **7** | 10.28 |
| 83004 | 1 | 1 | 1 | 2 | 802.00 | **732.89** | **106** | 0 | 9.73 |
| 75781 | 1 | 1 | 1 | **56** | 657.23 | **1808.42** | 2 | 0 | 8.29 |
| 75478 | 1 | 1 | 1 | **53** | 673.08 | **990.63** | 2 | 0 | 7.84 |
| 80710 | 1 | 1 | 1 | **50** | 435.46 | 6537.67 | 3 | **2** | 7.79 |
| 82623 | 1 | 1 | 1 | **48** | 365.35 | 8226.45 | 3 | 0 | 6.91 |
| 83918 | 1 | 1 | 1 | **47** | 452.85 | **2487.87** | 3 | 0 | 6.77 |

### 6.2.3 Experiment Case 3, SlowHeader Attack:

This experiment was performed on SlowHeader attack, which is one of the application layer attacks. This attack was performed in different scenarios as shown in section 4.2.1.2. we used Day3 normal dataset as the model centroid table and Day1,Day2 and Day4 were used for testing as shown in Table 6.9.

**Experiment Results**

The results of this experiment, shown in Table 6.10, shows that the detection rate of attack instances is 100% with 0.0% false positive rate at tune value equals 7.

**Table 6.9 BM-AUN2015 experiment case3, SlowHeader attack dataset**

| | Normal | | | | Attack | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Application layer attack | | | |
| | Day3 Centroid | Day1 | Day2 | Day4 | SlowHeader Scenario1 | SlowHeader Scenario2 | SlowHeader Scenario3 | SlowHeader Scenario4 |
| **#instances** | **13,548** | 703 | 3,296 | 10,332 | 2,949 | 2,980 | 218 | 213 |
| ∑ | **13,548** | **14,331** | | | **6,360** | | | |

Tune value 7 is the optimal tune value of this experiment where the true positive rate reaches 99.962% with false negative rate 0.038 % where as the true negative rate reaches 99.91% with false positive rate 0.00% as shown in Table 6.11.

Figure 6.3 shows an increase in normal instances accuracy labeling as we increase the tune value with a drop decrease in attack detection rate started at tune value equals 8.

**Table 6.10 BM-AUN2015 experiment case3, SlowHeader attack results**

| | Detection Rate | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Label/ Tune Value** | **0** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **12** | **30** |
| Normal | 0.843 | 0.971 | 0.977 | 0.978 | 0.978 | **0.999** | 0.999 | 0.999 | 0.999 | 0.999 | 1.000 |
| SlowHeader-Scenario1 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| SlowHeader-Scenario2 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 0.953 |
| SlowHeader-Scenario3 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | **1.000** | 0.995 | 0.922 | 0.601 | 0.069 | 0.000 |
| SlowHeader-Scenario4 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | **1.000** | 0.995 | 0.901 | 0.643 | 0.038 | 0.000 |

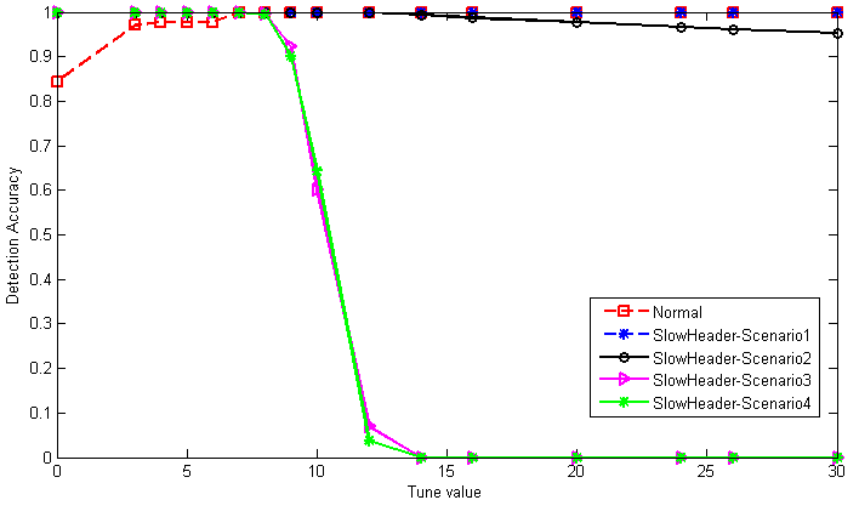


**Figure 6.3 BM-AUN2015 experiment case 3, SlowHeader attack results chart**

### Experiment Evaluation

Table 6.11 shows the confusion matrix results at the tune value 7 which is the optimal standard deviation expanding value with 99.99% correlation rate.

**Table 6.11 BM-AUN2015 experiment case3, SlowHeader attack confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 84.328 | 0.000 | 100.000 | 15.672 | 86.914 | 7.836 | 92.164 | 0.854 | 0.915 |
| 1 | 94.078 | 0.000 | 100.000 | 5.922 | 95.055 | 2.961 | 97.039 | 0.942 | 0.969 |
| 2 | 95.927 | 0.000 | 100.000 | 4.073 | 96.599 | 2.037 | 97.963 | 0.960 | 0.979 |
| 3 | 97.053 | 0.000 | 100.000 | 2.947 | 97.540 | 1.473 | 98.527 | 0.971 | 0.985 |
| 4 | 97.718 | 0.000 | 100.000 | 2.282 | 98.094 | 1.141 | 98.859 | 0.977 | 0.988 |
| 5 | 97.761 | 0.000 | 100.000 | 2.239 | 98.131 | 1.119 | 98.881 | 0.978 | 0.989 |
| 6 | 97.761 | 0.000 | 100.000 | 2.239 | 98.131 | 1.119 | 98.881 | 0.978 | 0.989 |
| **7** | **99.913** | **0.000** | **100.000** | **0.087** | **99.928** | **0.043** | **99.957** | **0.999** | **0.9996** |
| 8 | 99.913 | 0.232 | 99.768 | 0.087 | 99.921 | 0.159 | 99.841 | 0.997 | 0.998 |
| 9 | 99.928 | 4.414 | 95.586 | 0.072 | 99.807 | 2.243 | 97.757 | 0.957 | 0.979 |
| 10 | 99.928 | 18.897 | 81.103 | 0.072 | 99.369 | 9.485 | 90.515 | 0.835 | 0.920 |
| 16 | 99.957 | 50.310 | 49.690 | 0.043 | 98.361 | 25.177 | 74.823 | 0.489 | 0.832 |
| 30 | 99.986 | 51.166 | 48.834 | 0.014 | 98.128 | 25.590 | 74.410 | 0.484 | 0.828 |

Based on Table 6.11, the model has achieved 99.93% detection rate, 99.96% accuracy rate and a false positive rate 0.0% with a F-Measure rate reaches 100%.

**Experiment Results Discussion**

As shown in Table 6.12, the instances distance is so far from the normal class boundary which have a maximum distance 32.698. This is due to the extreme values of shaded cells features especially AVG_TIME_HTTP_HEADER_COMPLETE feature which is zero in normal instances and also due to AVG_TCP_PAYLOAD_LENGTH feature which is 377 bytes in average, as shown in Figure 4.31 and Figure 4.27 respectively. Our model shows that it has the ability to detect SlowHeader DDoS attacks.

**Table 6.12 BM-AUN2015 experiment case3, SlowHeader attack instances**

| The standard deviation = 2.698 + 7 = 9.698 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| NUMBER OF SERVER ACK | IS HTTP HEADER END | NUMBER OF CLIENT TCP PSH | AVG TCP PAYLOAD LENGTH | AVG CLNT TCP WINDOW SIZE | AVG CURRENT CONNECTIONS 4SEC | CLASS TYPE | NUMBER ZERO WINDOW PKTS | DISTANCE |
| 1 | 0 | 83 | 4 | 5687.05 | 361.66 | Scenario1 | 0 | 129.78 |
| 1 | 0 | 79 | 4 | 6436.24 | 404.71 | Scenario1 | 0 | 129.77 |
| 1 | 0 | 70 | 8 | 12621.69 | 345.02 | Scenario2 | 0 | 124.32 |
| 1 | 0 | 78 | 8 | 14204.37 | 343.37 | Scenario2 | 0 | 124.21 |
| 1 | 0 | 83 | 4 | 6431.65 | 5.00 | Scenario4 | 0 | 10.93 |
| 1 | 0 | 79 | 4 | 14073.81 | 20.00 | Scenario3 | 0 | 10.87 |
| 1 | 0 | 78 | 4 | 6044.69 | 19.69 | Scenario3 | 0 | 10.83 |

## 6.2.4 Experiment Case 4, Network Layer Attacks:

This experiment was performed on SYNC-Flood and SockStress attacks, which are one of the network layer attacks.. These attacks were performed in different scenarios as shown in section 4.2.1.2. We used Day3 normal dataset as the model centroid table and Day1,Day2 and Day4 were used for testing as shown in Table 6.13.

**Experiment Results**

The results of this experiment, shown in Table 6.14, shows that the detection rate of attack instances is 99.83% with 0.0% false positive rate and false negative rate = 4.91 at tune value .

**Table 6.13 BM-AUN2015 experiment case4, network layer attacks dataset**

| | Normal | | | | Attack | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Network layer attack | | | |
| | Day3 Centroid | Day1 | Day2 | Day4 | SockStress 20Thread | SockStress 40Thread | SYNC-FLOOD SingleIP | SYNC-FLOOD SpoofedIP |
| **#instances** | **13,548** | 703 | 3,296 | 10,332 | 1,929 | 6,876 | 65,540 | 462,206 |
| ∑ | **13,548** | **14,331** | | | **536,551** | | | |

The optimal tune value of this experiment is at tune value equals 12 where the true positive rate reaches 99.942% with false negative rate 0.057 % where as the true negative rate reaches 99.6% with false positive rate 0.398% as shown in Table 6.15.

Figure 6.4 shows an increase in normal instances accuracy labeling as we increase the tune value. The detection accuracy decreased in sockstress attack scenarios specially sockstress20thread where its detection accuracy start decreasing at tune value 3 until it suddenly decreased to 93.68% at tune value 30 whereas the other attacks stay approximately at a steady state.

**Table 6.14 BM-AUN2015 experiment case4, network layer attack results**

| | Detection Rate | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Label/Tune Value** | **0** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **12** | **16** | **30** |
| Normal | 0.712 | 0.969 | 0.979 | 0.980 | 0.980 | 0.980 | 0.999 | 0.999 | **0.999** | 0.999 | 0.9996 |
| SYNC-FLOOD-SpofedIP | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 |
| SYNC-FLOOD-SingleIP | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 |
| SockStress-40Thread | 1.0000 | 0.9999 | 0.9999 | 0.9999 | 0.9999 | 0.9999 | 0.9999 | 0.9999 | **0.9999** | 0.9962 | 0.9856 |
| SockStress-20Thread | 1.0000 | 0.9995 | 0.9995 | 0.9855 | 0.9855 | 0.9855 | 0.9855 | 0.9855 | **0.9855** | 0.9855 | 0.9368 |

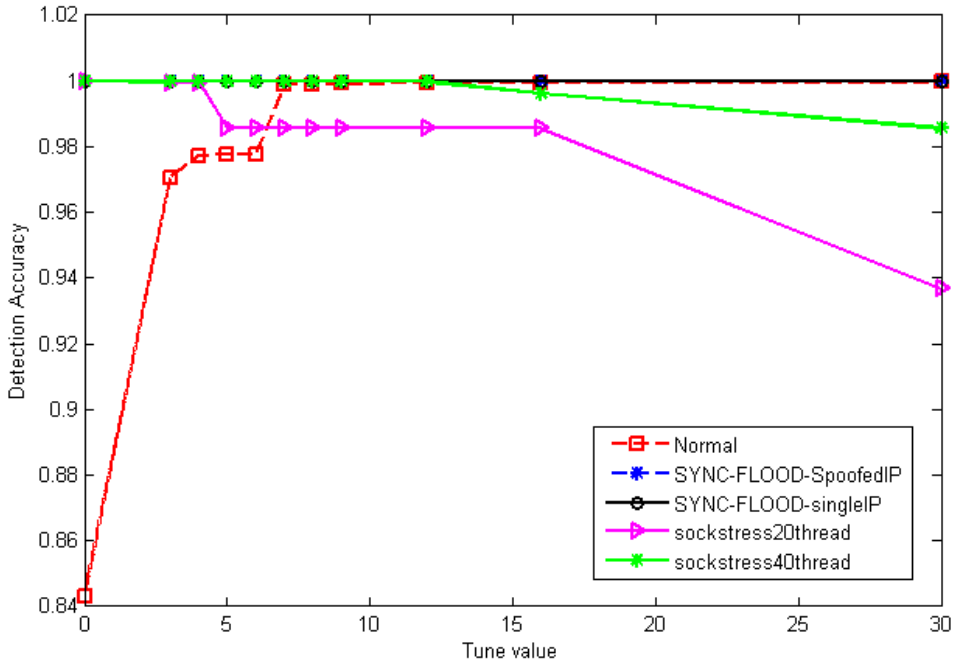


**Figure 6.4 BM-AUN2015 experiment case 4, network layer attack results chart**

**Experiment Evaluation**

Table 6.15 shows the confusion matrix results at the tune value 12 which is the optimal standard deviation expanding value. Based on Table 6.15, the model has achieved 99.87% detection rate, 99.77% accuracy rate and a false positive rate 0.399% with a correlation rate reaches 99.6% and F-Measure=99.77% .

**Table 6.15 BM-AUN2015 experiment case4, network layer attack confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 84.3276 | 0.0000 | 100.0000 | 15.6724 | 90.6076 | 7.8362 | 92.1638 | 0.8538 | 0.9150 |
| 1 | 94.0777 | 0.0000 | 100.0000 | 5.9223 | 96.4508 | 2.9611 | 97.0389 | 0.9424 | 0.9695 |
| 2 | 95.9266 | 0.0270 | 99.9730 | 4.0734 | 97.5483 | 2.0502 | 97.9498 | 0.9598 | 0.9791 |
| 3 | 97.0533 | 0.0490 | 99.9510 | 2.9467 | 98.2167 | 1.4979 | 98.5021 | 0.9705 | 0.9848 |
| 4 | 97.7178 | 0.0490 | 99.9510 | 2.2822 | 98.6148 | 1.1656 | 98.8344 | 0.9769 | 0.9882 |
| 5 | 97.7611 | 0.3989 | 99.6011 | 2.2389 | 98.5645 | 1.3189 | 98.6811 | 0.9738 | 0.9867 |
| 6 | 97.7611 | 0.3989 | 99.6011 | 2.2389 | 98.5645 | 1.3189 | 98.6811 | 0.9738 | 0.9867 |
| 7 | 99.9133 | 0.3989 | 99.6011 | 0.0867 | 99.8542 | 0.2428 | 99.7572 | 0.9952 | 0.9976 |
| 8 | 99.9133 | 0.3989 | 99.6011 | 0.0867 | 99.8542 | 0.2428 | 99.7572 | 0.9952 | 0.9976 |
| 9 | 99.9278 | 0.3989 | 99.6011 | 0.0722 | 99.8628 | 0.2356 | 99.7644 | 0.9953 | 0.9977 |
| **12** | **99.9422** | **0.3989** | **99.6011** | **0.0578** | **99.8715** | **0.2284** | **99.7716** | **0.9955** | **0.9977** |
| 16 | 99.9567 | 0.4898 | 99.5102 | 0.0433 | 99.8349 | 0.2666 | 99.7334 | 0.9947 | 0.9973 |
| 30 | 99.9856 | 1.9735 | 98.0265 | 0.0144 | 99.4544 | 0.9940 | 99.0060 | 0.9806 | 0.9903 |

**Experiment Results Discussion**

The HTTP OCC didn't train on network layer attacks, and despite of that it detected them in a high accuracy percentage. SYNC-Flood attacks specially with spoofed IP attack have been detected in 100% despite the increase in tune value which means that they have a distance so far from the boundary of normal class, note that the number of concurrent connections of this type of attacks is just one. On the other hand sockstress attack have been detected but there's 0.058% of their instances have been identified as normal. After investigation about these instances we found that there's 28 instance out of 1929 instances for sockstree20thread is labeled as normal and 1 instance out of 6876 for sockstress40thread is labeled as normal. They are labeled as normal because of the feature NUMBER_OF_SERVER_ACK which was 1. This happened because the packet capturing process was stopped without waiting the server to close all connections. We can conclude that the model can detect network attacks with a detection rate of 100%.

## 6.2.5 BM-AUN2015 HTTP OCC Overall Evaluation

On the previous experiment cases which was conducted on different types of attacks, we have provided different accuracy results based on the attack types to evaluate the model accuracy for each attack type. In this section we need to evaluate the overall accuracy of the BM-AUN2015 HTTP OCC model and determine the optimal tune value.

We have introduced different tune values for each experiment case. SlowRead attack experiment has an optimal tune value equals 7, SlowPost attack experiment has an optimal tune value equals 3.3, SlowHeader attack has an optimal tune value equals 7 and network attacks experiment has an optimal tune value equals 12. So the optimal tune value is the smallest one which is 3.3 at which most of attacks are detected in high detection rate, but also it has high false alarm rate as shown in Table 6.16 below. There's no comparisons with other models on BM-AUN2015 dataset because this dataset is a real dataset collected by us.

**Table 6.16 BM-AUN2015 HTTP OCC overall confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 84.328 | 0.000 | 100.000 | 15.672 | 87.830 | 8.502 | 92.164 | 0.854 | 0.915 |
| 2 | 95.927 | 0.007 | 99.993 | 4.073 | 96.834 | 2.083 | 97.960 | 0.960 | 0.979 |
| 3 | 97.053 | 0.045 | 99.955 | 2.947 | 97.707 | 1.518 | 98.504 | 0.970 | 0.985 |
| **3.3** | **97.617** | **0.133** | **99.867** | **2.383** | **98.141** | **1.275** | **98.742** | **0.975** | **0.987** |
| 3.8 | 97.703 | 0.477 | 99.523 | 2.297 | 98.197 | 1.411 | 98.613 | 0.972 | 0.986 |
| 6.3 | 98.570 | 1.806 | 98.194 | 1.430 | 98.812 | 1.701 | 98.382 | 0.969 | 0.984 |
| 7 | 99.913 | 2.605 | 97.395 | 0.087 | 99.828 | 1.487 | 98.654 | 0.975 | 0.988 |
| 9 | 99.928 | 7.906 | 92.094 | 0.072 | 99.687 | 4.702 | 96.011 | 0.928 | 0.965 |
| 12 | 99.942 | 27.815 | 72.185 | 0.058 | 99.083 | 25.039 | 86.063 | 0.753 | 0.901 |
| 30 | 99.986 | 38.866 | 61.134 | 0.014 | 98.527 | 38.216 | 80.560 | 0.608 | 0.868 |

As shown in Figure 6.5 the optimal tune value of BM-AUN2015 HTTP OCC is at tune =3.3

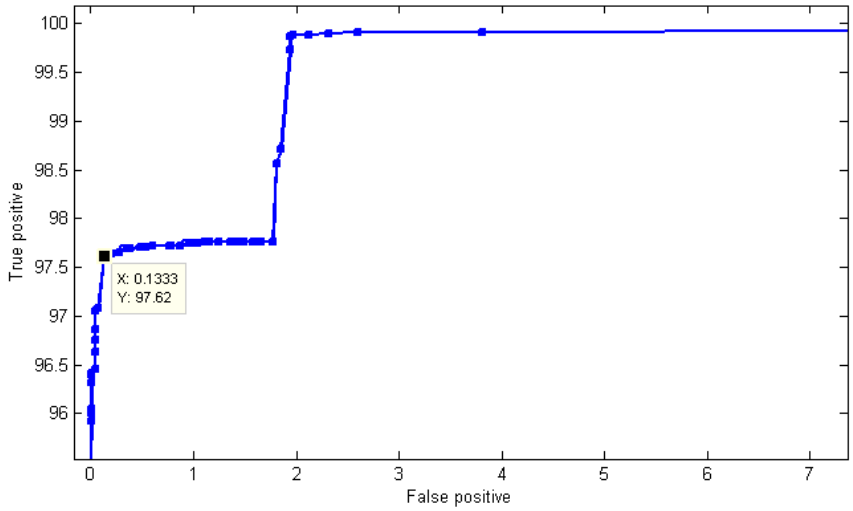


**Figure 6.5 BM-AUN2015 performance ROC curve**

## *6.3 KDD Cup'99 Experiments Cases and Results:*

Six experiment cases were conducted, two for each service, which are HTTP, ECR_I and POP3 services which were explained in Chapter 2. The OCC model of each service was built as described in section 5.4.1.2. The dataset used for each experiment is 10% testing KDD

Cup'99 dataset as shown in Table 5.10, 5.11, 5.12 for the services HTTP, ECR_I and POP3 respectively.

### 6.3.1 HTTP Service Experiment:

This experiment was performed on HTTP service dataset which was extracted from 10% testing KDD Cup'99 dataset as listed in Table 5.10. Table 5.10 shows the number of attack instances, attack type and attack name. The model was built using HTTP 10% training KDD Cup'99 dataset at tune value=6. MIT Lincoln Labs [20] provide a 10% corrected dataset from the full dataset. This dataset, as they described, is not from the same probability distribution as the training data, and it includes specific attack types not in the training data.

**Experiment Results**

The results of this experiment, shown in Table 6.17, achieved high detection accuracy with low false positive rate based on the optimal tune value of HTTP OCC model which equals 6. The true positive rate at tune value 6 reaches 99.3% with false negative rate 0.706 %, where as the true negative rate reaches 99.9950% with 0.0049% false positive rate as shown in Table 6.18.

**Table 6.17 KDD Cup'99 HTTP service testing experiment results**

| Label/Tune | Detection Rate | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **6** | **7** | **11** | **16** | **23** | **24** | **25** |
| normal. | 0.9175 | 0.9521 | 0.9799 | 0.9884 | 0.9911 | **0.9929** | 0.9934 | 0.9954 | 0.9961 | 0.9968 | 0.9969 | 0.9970 |
| apache2. | 0.9987 | 0.9962 | 0.9962 | 0.9962 | 0.9962 | **0.9962** | 0.9962 | 0.9836 | 0.9836 | 0.9761 | 0.9748 | 0.9748 |
| back. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| neptune. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| phf. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 0.5000 | 0.5000 | 0.5000 | 0.0000 | 0.0000 | 0.0000 |
| portsweep. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 0.9972 | 0.9689 | 0.9576 | 0.9576 | 0.9576 |
| saint. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |

Figure 6.6 shows an increase in normal instances detection accuracy as the tune value increased. The detection accuracy of three attacks of different types has been affected. These attacks are apache2 which is a DoS attack, the other is phf which is R2L attack and portsweep which is a probe attack. The detection accuracy of phf attack, as shown in Figure 6.6, decrased to 50% at tune value of 7. As described in chapter 2, R2L attacks are hard to be detected because of their near normal behaviour. The apache2 attack decreased with a relativly small amount.

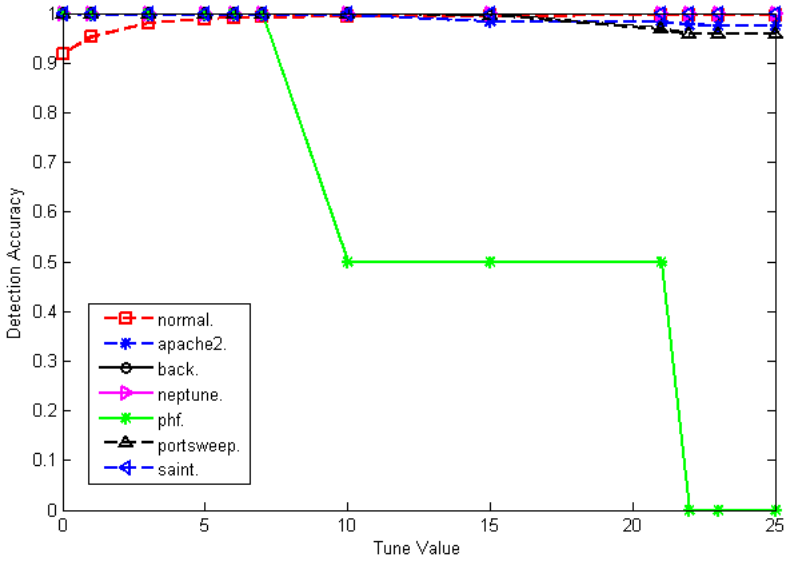


**Figure 6.6 KDD Cup'99 HTTP service experiment results chart**

### Experiment Evaluation

Table 6.18 shows the confusion matrix results at the tune value 6 which is the optimal standard deviation expanding value.

**Table 6.18 KDD Cup'99 HTTP experiment optimal confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 91.755 | 0.002 | 99.998 | 8.245 | 96.766 | 4.301 | 95.877 | 0.921 | 0.957 |
| 1 | 95.212 | 0.005 | 99.995 | 4.788 | 98.120 | 2.455 | 97.604 | 0.953 | 0.975 |
| 1.5 | 96.015 | 0.005 | 99.995 | 3.985 | 98.435 | 2.036 | 98.005 | 0.961 | 0.980 |
| 2 | 97.992 | 0.005 | 99.995 | 2.008 | 99.210 | 1.017 | 98.994 | 0.980 | 0.990 |
| 3 | 98.841 | 0.005 | 99.995 | 1.159 | 99.542 | 0.586 | 99.418 | 0.988 | 0.994 |
| 4 | 99.111 | 0.005 | 99.995 | 0.889 | 99.648 | 0.449 | 99.553 | 0.991 | 0.996 |
| 5 | 99.215 | 0.005 | 99.995 | 0.785 | 99.689 | 0.396 | 99.605 | 0.992 | 0.996 |
| **6** | **99.294** | **0.005** | **99.995** | **0.706** | **99.720** | **0.357** | **99.645** | **0.993** | **0.996** |
| 7 | 99.343 | 0.007 | 99.993 | 0.657 | 99.738 | 0.333 | 99.668 | 0.993 | 0.997 |
| 8 | 99.378 | 0.012 | 99.988 | 0.622 | 99.749 | 0.318 | 99.683 | 0.994 | 0.997 |
| 9 | 99.394 | 0.015 | 99.985 | 0.606 | 99.753 | 0.312 | 99.689 | 0.994 | 0.997 |
| 10 | 99.526 | 0.023 | 99.977 | 0.474 | 99.800 | 0.249 | 99.752 | 0.995 | 0.998 |
| 11 | 99.539 | 0.025 | 99.975 | 0.461 | 99.804 | 0.244 | 99.757 | 0.995 | 0.998 |
| 15 | 99.605 | 0.038 | 99.962 | 0.395 | 99.822 | 0.217 | 99.784 | 0.996 | 0.998 |

| Class | Attack | TN# | FP# | TN% | FP% |
|---|---|---|---|---|---|
| **DoS** | neptune. | 58001 | 0 | 100 | 0 |
| | apache2. | 791 | 3 | 99.62 | 0.379 |
| | back. | 1098 | 0 | 100 | 0 |
| **Probe** | saint. | 607 | 0 | 100 | 0 |
| | portsweep. | 354 | 0 | 100 | 0 |
| **R2L** | phf. | 2 | 0 | 100 | 0 |

Based on Table 6.18, our model has achieved 99.72% detection rate, 99.644% accuracy rate and a false positive rate 0.00493% with a correlation rate reaches 0.9929.

**Experiment Results Discussion**

The standard deviation of this HTTP OCC model is 3.78 which have been calculated based on experiment 8 shown in Table 5.17. The number of normal instances that are labeled as attack are 277 out of 39247 which were exceeded the expanded boundary value which is 3.78+6= 9.37. After investigation we found that the features that are responsible of this increase in distance are three features which are, src_bytes, num_compromised and dst_host_diff_srv_rate. These features contribute in the increasing distance because of their extreme values in these instances. As an example the normal average length of src_bytes in training dataset found is 238 but we found an existence of 23 instances that have a src_bytes value above 1000. Our model was perfect in detecting these extreme instances.
Our model prove that it is able to detect unknown attacks which doesn't exist in training dataset, see Table 5.10. These attacks are apache2 , a DoS attack, which has been 99.6% detected and saint attack, a probe attack, which has been 100% detected.

### 6.3.2 ECR_I Service Experiment:

We conducted two experiment cases. The first experiment was performed on ECR_I service dataset which was extracted from 10% testing KDD Cup'99 dataset as listed in Table 5.11. Table 5.11 shows the number of attack instances, attack type and attack name. The second experiment was performed on ECR_I service dataset extracted from the full KDD Cup'99 dataset which also listed in Table 5.11. The model was built using ECR_I 10% training KDD Cup'99 dataset at tune value=4.

***ECR_I OCC Model Validation using 10% testing KDD Cup'99 Dataset***

MIT Lincoln Labs [20] provide a 10% corrected dataset from the full dataset. This dataset, as they described, is not from the same probability distribution as the training data, and it includes specific attack types not in the training data.

**Experiment Results**

The results of this experiment, shown in Table 6.19, doesn't achieved a high detection accuracy of normal instances based on the optimal tune value of ECR_I OCC model which equals 4 even if we increase the tune value, the increment of detection accuracy of normal instances is relatively small . The true positive rate at tune value 4 reached 79.19% with false negative rate 20.809 %, where as the true negative rate reaches 99.938% with 0.061% false positive rate as shown in Table 6.20.

Based on the result of the base line experiment, shown in Table 5.22, the normal detection rate results of test experiment is unexpected and extremely differ than the base line experiment. We will apply this model to the complete dataset to validate our model accuracy after discussing the main problems that affect this result.

**Table 6.19 KDD Cup'99 ECR_I service testing experiment results**

| Label/TuneValue | Detection Rate | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **4** | **5** | **6** | **9** | **10** | **15** | **20** | **30** | **3000** |
| normal. | 0.6994 | 0.7919 | **0.7919** | 0.7919 | 0.7919 | 0.7919 | 0.7919 | 0.8150 | 0.8208 | 0.8208 | 0.9191 |
| ipsweep. | 1.0000 | 1.0000 | **1.0000** | 0.1797 | 0.1732 | 0.0131 | 0.0098 | 0.0098 | 0.0098 | 0.0098 | 0.0000 |
| pod. | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.0000 |
| saint. | 1.0000 | 1.0000 | **0.0098** | 0.0098 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| smurf. | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.0000 |

**Experiment Evaluation**

Table 6.20 shows the confusion matrix results at the tune value 4 which is the optimal standard deviation expanding value.

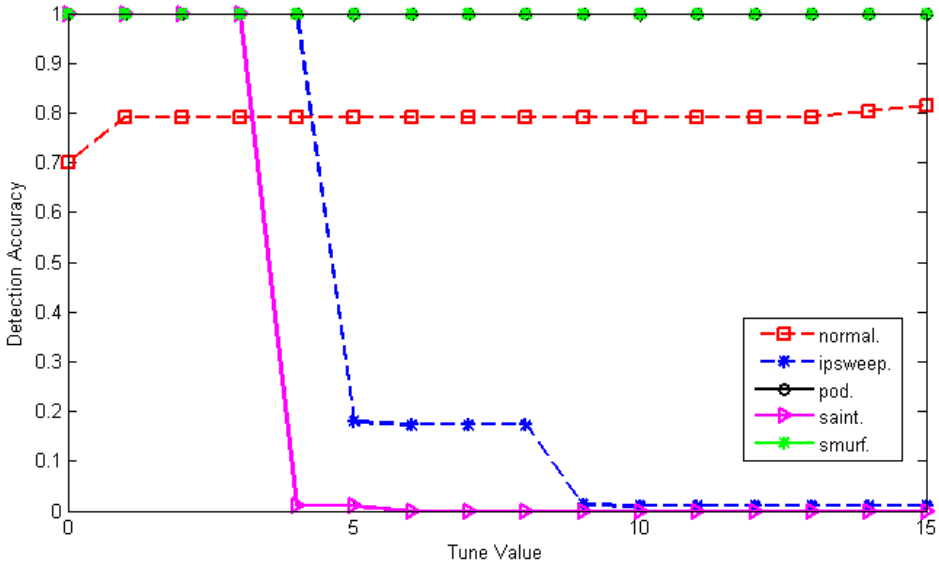


**Figure 6.7 KDD Cup'99 ECR_I service experiment results chart**

Based on Table 6.20, our model has achieved 99.916% detection rate, 89.6% accuracy rate and a false positive rate 0.061% with a correlation rate reaches 88.4%.

**Table 6.20 KDD Cup'99 ECR_I experiment optimal confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 69.942 | 0.000 | 100.000 | 30.058 | 99.968 | 17.687 | 84.971 | 0.733 | 0.823 |
| 3 | 79.191 | 0.000 | 100.000 | 20.809 | 99.978 | 11.613 | 89.595 | 0.810 | 0.884 |
| **4** | **79.191** | **0.061** | **99.939** | **20.809** | **99.917** | **11.651** | **89.565** | **0.809** | **0.884** |
| 5 | 79.191 | 0.214 | 99.786 | 20.809 | 99.764 | 11.746 | 89.488 | 0.807 | 0.883 |
| 7 | 79.191 | 0.216 | 99.784 | 20.809 | 99.763 | 11.747 | 89.488 | 0.807 | 0.883 |
| 15 | 81.503 | 0.246 | 99.754 | 18.497 | 99.735 | 10.341 | 90.628 | 0.826 | 0.897 |
| 20 | 82.081 | 0.246 | 99.754 | 17.919 | 99.735 | 9.990 | 90.917 | 0.831 | 0.900 |
| 30 | 82.081 | 0.246 | 99.754 | 17.919 | 99.735 | 9.990 | 90.917 | 0.831 | 0.900 |
| 100 | 82.081 | 0.246 | 99.754 | 17.919 | 99.735 | 9.990 | 90.917 | 0.831 | 0.900 |

| Class | Attack | TN# | FP# | TN% | FP% |
|---|---|---|---|---|---|
| DoS | smurf. | 164091 | 0 | 100 | 0 |
| | pod. | 81 | 0 | 100 | 0 |
| Probe | ipsweep. | 306 | 0 | 100 | 0 |
| | saint. | 1 | 101 | 100 | 0 |

**Experiment Results Discussion**

Figure 6.7 shows a very slow increment of normal instances' detection accuracy even we move far away from the normal class boundaries. After investigation from the testing dataset results we found that 36 out of 173 instances are labeled as attack. The main contributor of this problem was the src_bytes feature. The average length of src_bytes feature found in normal class is 32 but we found 37 instances that have more than the double of normal length. We found 31 instances have src_bytes value above 1480 and the rest which are 5 instances have src_bytes value above 64. This could happen only by the network administrator for network diagnostic operations.

***ECR_I OCC Model Validation using Full KDD Cup'99 Dataset***

In order to validate the robustness of KDD Cup'99 ECR_I OCC model, we need to test it on the full ECR_I dataset listed in Table 5.11. Table 5.11 shows the number of attack instances, attack type and attack name. The model was built using ECR_I 10% training KDD Cup'99 dataset.

**Experiment Results**

The results of this experiment, shown in Table 6.21, achieved high detection accuracy of both normal and attack instances at tune value equals 4. The true positive rate at tune value 4 reached 98.408% with false negative rate 1.591% where as the true negative rate reaches 99.999% with 0.00074% false positive rate as shown in Table 6.22.

Figure 6.8 shows an increase in normal instances detection accuracy as we increase the tune value. Whereas three Prope attacks' detection accuracy were decrease as we increase the tune value. Satan attack appears to be hard to be detected by the model, its detection accuracy droped extremely at a tune value 4. The second Prope attack, ipsweep, was little hard than satan attack to be detected by the model, its detection accuracy droped extremely at a tune value 5. The last Prope attack which is nmap attack was relatively weak, its detection accuracy also droped extremely to zero at tune value 9 which is far away from the model optimal tune value which equals to 4.

**Table 6.21 KDD Cup'99 full ECR_I service dataset testing experiment results**

| Label/TuneValue | Detection Rate | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 9 | 10 | 15 |
| normal. | 0.8715 | 0.9447 | 0.9592 | 0.9711 | **0.9841** | 0.9841 | 0.9841 | 0.9841 | 0.9841 | 1.0000 |
| ipsweep. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 0.7085 | 0.7085 | 0.0000 | 0.0000 | 0.0000 |
| nmap. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 0.0000 | 0.0000 | 0.0000 |
| pod. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| portsweep. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 0.6667 | 0.0000 | 0.0000 |
| satan. | 1.0000 | 1.0000 | 0.9189 | 0.9189 | **0.4324** | 0.2703 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| smurf. | 1.0000 | 1.0000 | 1.0000 | 1.0000 | **1.0000** | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |

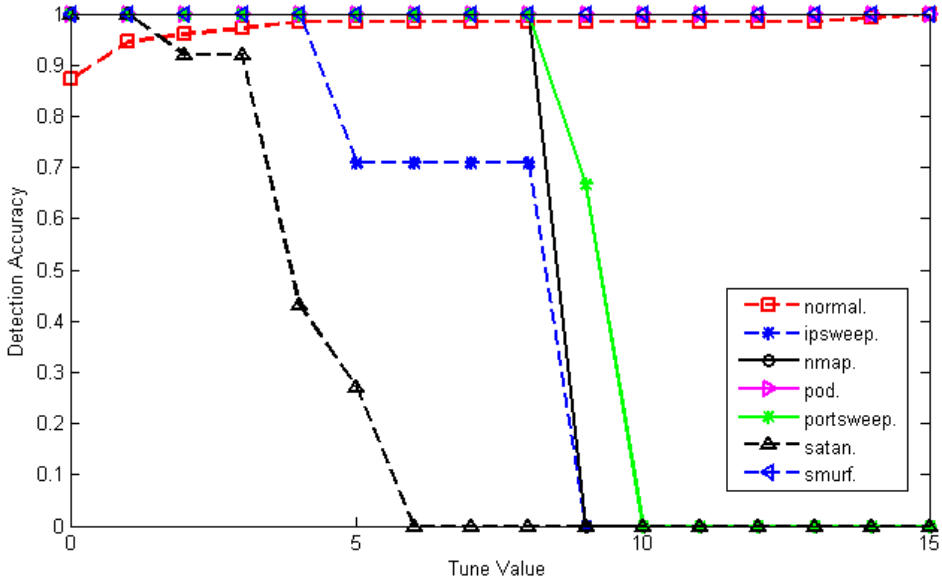


**Figure 6.8 KDD Cup'99 full ECR_I service dataset experiment results chart**

### Experiment Evaluation

Table 6.22 shows the confusion matrix results at the tune value 4 which is the optimal standard deviation expanding value. Based on Table 6.22, our model has achieved 99.997% detection rate, 99.203% accuracy rate and a false positive rate 0.00074% with a correlation rate reached 0.992%.

### Experiment Results Discussion

As shown in Figure 6.8, ECR_I OCC on full dataset shows that it can achieved a high detection accuracy rate of both normal and attack instances. After investigation about the false negative instances from the full ECR_I dataset results we found that 55 out of 3401 are labeled as attack.

**Table 6.22 KDD Cup'99 full ECR_I dataset experiment optimal confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 87.153 | 0.000 | 100.000 | 12.847 | 99.984 | 6.865 | 93.576 | 0.879 | 0.931 |
| 1 | 94.473 | 0.000 | 100.000 | 5.527 | 99.993 | 2.842 | 97.237 | 0.946 | 0.972 |
| 2 | 95.920 | 0.000 | 100.000 | 4.080 | 99.995 | 2.082 | 97.960 | 0.960 | 0.979 |
| 3 | 97.106 | 0.000 | 100.000 | 2.894 | 99.996 | 1.468 | 98.553 | 0.971 | 0.985 |
| **4** | **98.409** | **0.001** | **99.999** | **1.591** | **99.997** | **0.802** | **99.204** | **0.984** | **0.992** |
| 5 | 98.409 | 0.120 | 99.880 | 1.591 | 99.878 | 0.863 | 99.144 | 0.983 | 0.991 |
| 6 | 98.409 | 0.121 | 99.879 | 1.591 | 99.877 | 0.863 | 99.144 | 0.983 | 0.991 |
| 7 | 98.409 | 0.121 | 99.879 | 1.591 | 99.877 | 0.863 | 99.144 | 0.983 | 0.991 |
| 8 | 98.409 | 0.121 | 99.879 | 1.591 | 99.877 | 0.863 | 99.144 | 0.983 | 0.991 |
| 9 | 98.409 | 0.448 | 99.552 | 1.591 | 99.551 | 1.030 | 98.980 | 0.980 | 0.990 |
| 10 | 98.409 | 0.448 | 99.552 | 1.591 | 99.551 | 1.030 | 98.980 | 0.980 | 0.990 |
| 11 | 98.409 | 0.448 | 99.552 | 1.591 | 99.551 | 1.030 | 98.980 | 0.980 | 0.990 |
| 12 | 98.409 | 0.448 | 99.552 | 1.591 | 99.551 | 1.030 | 98.980 | 0.980 | 0.990 |

| Class | Attack | TN# | FP# | TN% | FP% |
|---|---|---|---|---|---|
| **DoS** | smurf. | 2807886 | 0 | 100 | 0 |
| | pod.. | 259 | 0 | 100 | 0 |
| **Probe** | ipsweep. | 11557 | 0 | 100 | 0 |
| | portsweep. | 6 | 0 | 100 | 0 |
| | satan. | 16 | 21 | 43.24 | 56.76 |
| | nmap. | 1032 | 0 | 100 | 0 |

The main contributor of this problem was the same as we described before in 10% ECR_I testing dataset experiment results discussion. These extreme instances are rare to happen because the src_bytes of ECR_I service, as described in chapter 2, is in general relatively small compared with the 1.59% instances that have a large src_bytes value. This is normally happens by network administrator just for testing purposes. Our model also detected unknown probe attack that doesn't exists in training dataset which is satan in a low percentage 43.24% detection rate.

### 6.3.3 POP3 Service Experiment:

The experiment was performed on POP3 service dataset which was extracted from 10% testing KDD Cup'99 dataset as listed in Table 5.12. Table 5.12 shows the number of attack instances, attack type and attack name. The model was built using POP3 10% training KDD Cup'99 dataset at tune value=1. MIT Lincoln Labs [20] provide a 10% corrected dataset from the full dataset. This dataset, as they described, is not from the same probability distribution as the training data, and it includes specific attack types not in the training data.

#### Experiment Results

The results of this experiment, shown in Table 6.23, achieved high detection accuracy with low false positive rate based on the optimal tune value of POP3 OCC model which equals 1.

The true positive rate at tune value 1 reacheD100% with false negative rate 0.0 %, where as the true negative rate reached 99.923% with 0.0765% false positive rate as shown in Table 6.24.

**Table 6.23 KDD Cup'99 POP3 service testing experiment results**

| Label/TuneValue | Detection Rate | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **1.5** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| normal. | 0.933 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| guess_passwd. | 0.995 | **0.990** | 0.989 | 0.988 | 0.987 | 0.986 | 0.985 | 0.985 | 0.984 | 0.984 |
| mscan. | 1.000 | **0.989** | 0.977 | 0.971 | 0.95 | 0.934 | 0.914 | 0.896 | 0.880 | 0.856 |
| neptune. | 1.000 | **1.000** | 1.000 | 1.000 | 1.000 | 1.000 | 0.999 | 0.999 | 0.999 | 0.999 |

Figure 6.9 shows an increase in normal instances detection accuracy as we increase the tune value. The detection accuracy of two attacks of different types has been affected. These attacks are guess_passwd which is R2L attack and the other is mscan attack which is Probe attack. The detection accuracy of mscan attack deceased relatively rapidly compared with gues_passwd attack . As described in chapter 2, Probe attacks are hard to be detected because of their near normal behaviour. The guess_passwd attack decreased with a relativly small amount as we increase the tune value.

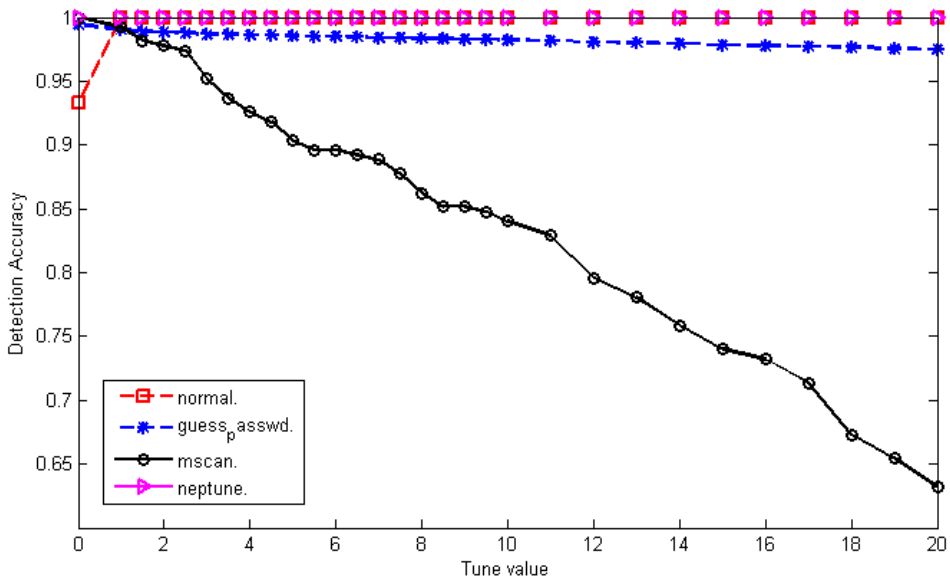


**Figure 6.9 KDD Cup'99 POP3 service experiment results chart**

### Experiment Evaluation

Table 6.24 shows the confusion matrix results at the tune value 1 which is the optimal standard deviation expanding value. Based on Table 6.24, our model has achieved 99.923% detection rate, 99.962% accuracy rate and a false positive rate 0.0765% with a correlation rate reaches 0.9992 at tune value 1.

**Table 6.24 KDD Cup'99 POP3 experiment optimal confusion matrix results**

| TUNE | TPR | FPR | TNR | FNR | Detection% | Class. Err. | Accuracy | Correl. | F-Measure |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 93.333 | 0.032 | 99.968 | 6.667 | 99.967 | 3.465 | 96.651 | 0.935 | 0.965 |
| **1** | **100.000** | **0.077** | **99.923** | **0.000** | **99.923** | **0.038** | **99.962** | **0.999** | **1.000** |
| 1.5 | 100.000 | 0.104 | 99.896 | 0.000 | 99.896 | 0.052 | 99.948 | 0.999 | 0.999 |
| 2 | 100.000 | 0.120 | 99.880 | 0.000 | 99.880 | 0.060 | 99.940 | 0.999 | 0.999 |
| 3 | 100.000 | 0.164 | 99.836 | 0.000 | 99.836 | 0.082 | 99.918 | 0.998 | 0.999 |
| 4 | 100.000 | 0.195 | 99.805 | 0.000 | 99.805 | 0.097 | 99.903 | 0.998 | 0.999 |
| 5 | 100.000 | 0.234 | 99.766 | 0.000 | 99.766 | 0.117 | 99.883 | 0.998 | 0.999 |
| 6 | 100.000 | 0.268 | 99.732 | 0.000 | 99.732 | 0.134 | 99.866 | 0.997 | 0.999 |
| 7 | 100.000 | 0.298 | 99.702 | 0.000 | 99.702 | 0.149 | 99.851 | 0.997 | 0.999 |
| 8 | 100.000 | 0.343 | 99.657 | 0.000 | 99.657 | 0.172 | 99.829 | 0.997 | 0.998 |
| 9 | 100.000 | 0.375 | 99.625 | 0.000 | 99.625 | 0.188 | 99.813 | 0.996 | 0.998 |

| Class | | TN# | FP# | TN% | FP% |
|---|---|---|---|---|---|
| DoS | neptune. | 58000 | 1 | 99.998 | 0.0017 |
| Probe | mscan. | 1041 | 12 | 98.86 | 1.1396 |
| R2L | guess_passwd. | 3607 | 35 | 99.038 | 0.962 |

**Experiment Results Discussion**

Compared with the baseline experiment shown in Table 5.25, the test experiment achieved 100% high positive rate, where in baseline experiment it is 97.2%. In the baseline experiment exist 2 instances out of 77 instances identified as attacks. These two instances have distances 6.7 and 7.22 which is so far from the boundary of the OCC normal class based on the model standard deviation shown in Table 5.23, which is 2.236 +1 tune value = 3.236. Two features have a contribution of this increase which are dst_host_srv_serror_rate and count_v which have extreme values compared with the normal instances. These two features have high values in neptune attack. But in the testing dataset, there's no existence of such these instances. Our model prove its ability on detecting unknown attack which doesn't exists in the training dataset these attacks are guess_pass. which is R2L attack that detected in 99.038% and mscan. which is a probe attack that detected in 98.86%.

## *6.4 Running Time*

The experiments were conducted using an Intel® Core™2 Duo CPU 1.8GHz with 2.5GB RAM. The time consumed for the three services of KDD Cup'99 , where are HTTP, POP3 and ECR_I , are differs depends on the number of features that included in the distances measurement. Table 6.25 listed the number of instances, average time consumed for all instance of a service and the number of features of each service. Note that the number of features of HTTP and POP3 services, shown in Table 5.16, 5.22 respectively, are differs than their number of features at the running time because of converting categorical and binary features into multiple features, these features are flag, logged_in features.

**Table 6.25 KDD Cup'99 OCC Models running time**

| | # Instances | Avg time(Sec) | Time/Instance (Micro-Sec) | # Features |
|---|---|---|---|---|
| HTTP | 102,459 | 8.49 | 82.83 | 27 |
| POP3 | 64,975 | 3.21 | 49.33 | 22 |
| ECR_I | 164,754 | 5.86 | 35.58 | 6 |

As shown in Table 6.25, as number of features increased, the execution time is also increased.

## *6.5 Comparison with Other Models*

Based on our experiments that were carried out using KDD Cup'99. We chose only three services which are HTTP, POP3 and ECR_I services because of the existence of enough normal instances and varying types of attacks that exploited these services, but we still didn't represent the overall attacks exists in KDD Cup'99 dataset. Because of this limitation we decide to compare our results based on the attack type detection rate with the available related work models that present their experiment results based on attack type detection rate. We chose five models that apply different data mining learning techniques using supervised, unsupervised, semi-supervised and OCC learning techniques.

As shown in Table 6.26, we achieved higher detection rates in all DoS attacks, specially apache2 attach.

**Table 6.26 Comparison with other models**

| Model | normal | neptune | Smurf | pod | back | apache2 | Approach |
|---|---|---|---|---|---|---|---|
| Our Model | 99.26 | 99.998 | 100 | 100 | 100 | 99.6 | OCC-Service |
| Almutairi [66] | 99.3 | 99.98 | 99.8 | 98.7 | 98.4 | N/A | Naïve-Bayes |
| Yang [77] | 99.3 | 99.97 | 100 | 98.85 | 99.36 | 58.94 | Classification/Clustering |
| Jiang [50] | 99.83 | 100 | 99.89 | 18.18 | 3.29 | N/A | Clustering/Outlier factor |
| Abd-Eldayem [70] | 99.4 | 100 | N/A | N/A | 100 | 95 | HTTP- Naïve-Bayes |
| Barhoom and Matar [16] | 98.84 | 100 | 100 | 98.9 | 99 | 97.5 | OCC-Transport Protocols |

The bar chart of this comparison is shown in Figure 6.10. We have thre DoS attacks that are detected in 100% detection rate while other models don't reached this detection rate. This is because we accept relatively high false alarm in order to not miss R2L and probe attacks and to detect all attacks in a high detection rate. Note that as we increase the tune value the false alarm is decreased whereas the attack detection rate is decreased, specially R2L and Probe attacks.

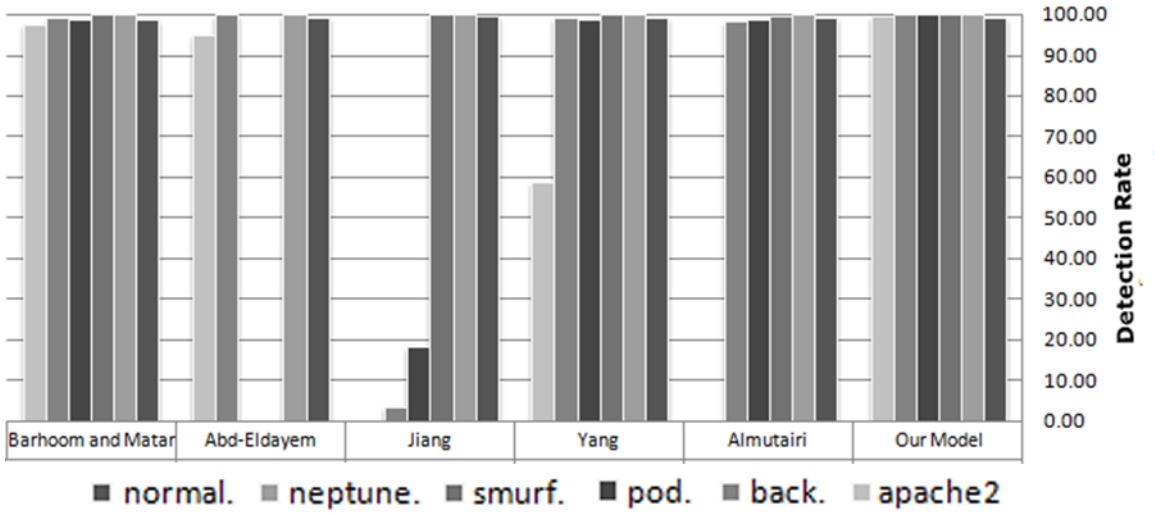


**Figure 6.10 Comparison with other models chart**

## *6.5 Summary*

Many experiments was performed on two datasets, BM-AUN2015 and KDD Cup'99. We performed four experiments on BM-AUN2015 dataset, three for application layer attacks and one for network layer attacks. Each experiment has four scenarios, two scenarios for DoS attacks and two scenarios for DDoS attacks. Also we performed four experiments on KDD Cup'99, one experiment for HTTP service, one experiment for POP3 service and two experiments for ECR_I service, one experiment on 10% ECR_I dataset and the other experiment was performed using the full ECR_I dataset. We evaluate and discussed each experiment results.

The results showed that the proposed model has achieved a higher accuracy and detection rate with low false alarm and low false positive rates as shown in Table 6.27.

**Table 6.27 OCC Models performance summary**

| Dataset | Detection% | Accuracy% | False Positive% | False Alarm% |
|---|---|---|---|---|
| SlowRead | 99.917% | 99.940% | 0.033% | 0.087% |
| SlowPost | 97.999% | 98.566% | 0.484% | 2.383% |
| SlowHeader | 99.928% | 99.957% | 0.0% | 0.087% |
| NetworkLayer | 99.872% | 99.772% | 0.399% | 0.057% |
| **BM-AUN2015 HTTP** | **98.141%** | **98.742%** | **0.133%** | **2.383%** |
| KDD Cup'99 HTTP 10% | 99.299% | 99.616% | 0.063% | 0.706% |
| KDD Cup'99 ECR_I 10% | 86.251% | 89.595% | 0.0% | 20.809% |
| KDD Cup'99 ECR_I 100% | 97.630% | 98.553% | 0.0% | 2.894% |
| KDD Cup'99 POP3 10% | 99.446% | 99.716% | 0.568% | 0.0% |

As shown in Table 6.27 there's four OCC models, an HTTP OCC model was built using BM-AUN2015 dataset and three services which are HTTP OCC, ECR_I OCC, and POP3 OCC, were built using KDD Cup'99 10% dataset except ECR_I OCC model which has two OCC,

the first was built on KDD Cup'99 10% dataset and the second was built using full KDD Cup'99 dataset for evaluation.

The SlowPost-DDoS attack, as shown in Table 6.27 and Table 6.6, has low detection rate, 97.999%, and high false alarm rate, 2.38% . This gives us an indication that the SlowPost-DDoS attack appeared to be hard for detection. This is because that 2.3% cases of normal instances have the same pattern like SlowPost attack as described subsection 6.2.2.

In KDD Cup'99 ECR_I 10%, we found that , as shown in Table 6.27 and Table 6.19, this service has low detection rate, 86.251%, and high false alarm rate, 20.809%. We explained the main problem of this in subsection 6.3.2. The model evaluation of this service was carried out using the full dataset, which achieved relatively high detection rate and low false alarm.

We observed from the results of training phase and testing phase that our model was very robust against DoS attacks and perform well with Probe attacks although that the Probe attacks are a stealthy attacks which are hard to be detected.

The limitation of our primary model is the determination of most relevant features of a certain service. In chapter 5 we used the training datasets, BM-AUN2015 and KDD Cup'99, that include attacks and normal instances in order to select the optimal features set which gives high accuracy rate. But in a real environment these datasets are not available.

# Chapter 7: Conclusion and Future work

This chapter concludes the work, its results and discussion. The future work directions were remarked.

## *7.1 Discussion and Summary:*

These days, abnormal network traffic is a critical threat on computer network. There are several researches have been proposed to manipulate this problem. In this research, we proposed an efficient model using OCC technique based on the standard deviation of service's normal behavior. Through this model we dealt with each network service as single class instead of dealing with all network services as a single class. By this way we use just the relevant features of each service, hence reducing the high dimensional network feature spaces and also ensure that each class has - a proximately - uniform distribution. The instance is considered to be labeled as Abnormal if the distance between the new instance and its relevant service's centroid table is greater than the service's class standard deviation, else it as labeled as Normal. The model consists of three main phases:

**Phase 1, Dataset collection and preparation:** To build and evaluate our model we used two datasets. KDD Cup '99 dataset [20] was used. Three services, HTTP, POP3 and ECR_I, were extracted from this dataset, the most relevant features for each service were selected. Another dataset, called BM-AUN2015 was used. This dataset is a real dataset which was collected from the traffic came from and going to Alaqsa University web server. For more information about this dataset please refer to chapter 4.

**Phase 2, Building OCC model:** In this phase, we used the predefined datasets from phase1. Four datasets were in hand, which are BM-AUN2015, KDD Cup'99 HTTP, KDD Cup'99 POP3, and KDD Cup'99 ECR_I. The normal instances were extracted from datasets and were preprocessed separately, e.g. sampling, over-sampling, outlier elimination, categorical features conversion and z-score normalization. Each normal dataset was dealt as a the class of the OCC model. A centroid table was generated for each class. Based on the generated centroid tables, the standard deviation of each class was obtained. Multiple experiments for each dataset were performed. The purpose of these experiments is to select the most relevant features for each service's class. Based on the optimal features set of each class, a baseline experiments for each class were conducted and the optimal tune value was determined.

**Phase 3, OCC model evaluation:** In this phase, we performed four experiments on BM-AUN2015 dataset, three for application layer attacks and one for network layer attacks. Each experiment has four scenarios, two scenarios for DoS attacks and two scenarios for DDoS attacks. Also we performed four experiments on KDD Cup'99, one experiment for HTTP service, one experiment for POP3 service and two experiments for ECR_I service. We evaluate and discussed each experiment results.

The results showed that the proposed model has achieved a higher accuracy and detection rate with low false alarm and low false positive rates. As shown in Table 7.1, BM-AUN2015 HTTP OCC achieved 98.14% detection rate, 98.74 accuracy rate, false positive rate 0.133%, and 2.38% false alarm rate. While KDD Cup'99 HTTP OCC achieved 99.299% detection rate, 99.616%, with false positive rate 0.063% and false alarm rate 0.706%. POP3 service in KDD Cup'99 achieved 99.446% detection rate, and 99.716% accuracy rate, with 0.568% false positive rate and 0.0% false alarm rate.

**Table 7.1 OCC Models performance / Service**

| Dataset | Detection% | Accuracy% | False Positive% | False Alarm% |
|---|---|---|---|---|
| BM-AUN2015 HTTP | 98.141% | 98.742% | 0.133% | 2.383% |
| KDD Cup'99 HTTP 10% | 99.299% | 99.616% | 0.063% | 0.706% |
| KDD Cup'99 ECR_I 10% | 86.251% | 89.595% | 0.0% | 20.809% |
| **KDD Cup'99 ECR_I 100%** | **97.630%** | **98.553%** | **0.0%** | **2.894%** |
| KDD Cup'99 POP3 10% | 99.446% | 99.716% | 0.568% | 0.0% |

We observed that ECR_I service in KDD Cup'99 10% didn't achieved an acceptable detection rate due to some normal instances that have similar pattern as attack instances, these normal instances, as described in subsection 6.3.2, could be happened by network administrator as a diagnosis operations. We evaluate the ECR_I OCC model using KDD Cup'99 full dataset which prove that these extreme normal instances are rare to be happen. As shown in Table 7.1 ECR_I OCC using complete KDD Cup'99 dataset achieved 97.63% detection rate and 98.554% accuracy rate were the false positive rate was 0% and false alarm rate was 2.89%.

## *7.2 Future Work*

- Find a suitable One-Class features selection method based on the normal instances.
- Evaluate OCC based on the standard deviation of normal behavior on content based attacks such as SQL Injection.
- Evaluate OCC based on the standard deviation of normal behavior on worms.
- Evaluate the OCC based on the standard deviation of normal behavior on applications other than IDS such as detection computer viruses.

## 10. References


[1] ISC, " ISC Internet domain survey (January 2015)." *Internet Systems Consortium, Inc . http://ftp.isc.org/www/survey/reports/2015/01. Accessd on: 13/02/2015,* 2015.

[2] S. S. Greene, *Security policies and procedures*: New Jersey: Pearson Education, 2006.

[3] S. Ben-David, T. Lu, and D. Pál, "Does Unlabeled Data Provably Help? Worst-case Analysis of the Sample Complexity of Semi-Supervised Learning," in *COLT*, 2008, pp. 33-44.

[4] J. McHugh, "Intrusion and intrusion detection," *International Journal of Information Security,* vol. 1, pp. 14-35, 2001.

[5] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *IEEE Security & Privacy,* vol. 1, pp. 33-39, 2003.

[6] J. P. Anderson, "Computer security threat monitoring and surveillance," Technical report, James P. Anderson Company, Fort Washington, Pennsylvania1980.

[7] M. A. Bishop, "The art and science of computer security," *Addison-Wesley Longman Publishing Co., Inc. Boston, MA,USA,* 2002.

[8] R. Heady, G. F. Luger, A. Maccabe, and M. Servilla, *The architecture of a network level intrusion detection system*: Department of Computer Science, College of Engineering, University of New Mexico, 1990.

[9] C. Shields, "Machine Learning and Data Mining for Computer Security, chapter An Introduction to Information Assurance," ed: Springer, 2005.

[10] A. P. R. da Silva, M. H. Martins, B. P. Rocha, A. A. Loureiro, L. B. Ruiz, and H. C. Wong, "Decentralized intrusion detection in wireless sensor networks," in *Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks*, 2005, pp. 16-23.

[11] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks,* vol. 31, pp. 805-822, 1999.

[12] BLAND, J.M.A., and D.G., "STATISTICS NOTES: MEASUREMENT ERROR," 1996.

[13] M. M. Moya and D. R. Hush, "Network constraints and multi-objective optimization for one-class classification," *Neural Networks,* vol. 9, pp. 463-474, 1996.

[14] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Artificial Intelligence and Cognitive Science*, ed: Springer, 2010, pp. 188-197.

[15] Tax D. and Duin R., *Uniform object generation for optimizing one-class classifiers.*: J. Machine Learning Research 2, 2001.

[16] T. S. Barhoom and R. A. Matar, "Network Intrusion Detection Using Semi-Supervised Learning Based on Normal Behaviour's Standard Deviation," *Network,* vol. 4, 2015.

[17] E. Shi and A. Perrig, "Designing secure sensor networks," *Wireless Communications, IEEE,* vol. 11, pp. 38-43, 2004.

[18] S. Kumar and E. H. Spafford, "A pattern matching model for misuse intrusion detection," 1994.

[19] J. Hochberg, K. Jackson, C. Stallings, J. McClary, D. DuBois, and J. Ford, "NADIR: An automated system for detecting network intrusion and misuse," *computers & security,* vol. 12, pp. 235-248, 1993.

[20] KDD, "The third international knowledge discovery and data mining tools competition dataset (KDD99 Cup). http://kdd.ics.uci.edu/databases/kddcup99/ ; Accessed on: 24/12/2014," 1999.

[21] Heady R, Luger G, Maccabe A, and Servilla M, "The Architecture of a Network Level Network Intrusion Detection System. (Technical Report CS90-20) " *University of New Mexico: Department of Computer Science,* 1990.

[22] D. Hollingworth, "Towards threat, attack, and vulnerability taxonomies," in *IFIP WG 10.4 Meeting. Monterey*, CA, USA, 2003.

[23] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks,* vol. 34, pp. 579-595, 2000.

[24] Kharat J.S. and Radhakrishna Naik, "A VIVACIOUS APPROACH TO DETECT AND PREVENT DDoS ATTACK," *International Journal of Research in Engineering and Technology,* vol. Vol. 02 2013.

[25] Pavithra K.C, Snitha Shetty, and Nagesh H.R, "A COMPREHENSIVE STUDY ON DISTRIBUTED DENIAL OF SERVICE ATTACKS AND DEFENCE MECHANISMS," *IJCA Proceedings on International Conference on Information and Communication Technologies,* 2014.

[26] J. Yu, C. Fang, L. Lu, and Z. Li, "Mitigating application layer distributed denial of service attacks via effective trust management," *IET communications,* vol. 4, pp. 1952-1962, 2010.

[27] H. Beitollahi and G. Deconinck, "Tackling application-layer DDoS attacks," *Procedia Computer Science,* vol. 10, pp. 432-441, 2012.

[28] Felix Lau, Stuart H. Rubin, Michael H. Smith, and Lj ilj ana Traj koviC, "Distributed Denial of Service Attacks," *IEEE,* 2000.

[29] Yuichi Ohsita, Shingo Ata, and Masayuki Murata, "Detecting Distributed Denial-of-Service Attacks by analyzing TCP SYN packets statistically," *IEEE Communications Society,* 2004.

[30] J. Hutchens, *Kali Linux Network Scanning Cookbook*: Packt Publishing Ltd, 2014.

[31] Saman Taghavi Zargar, James Joshi, and David Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE COMMUNICATIONS SURVEYS,* vol. Vol. 15, No. 4, FOURTH QUARTER 2013 2013.

[32] T. S. Sobh, "Wired and wireless intrusion detection system: Classifications, good characteristics and state-of-the-art," *Computer Standards & Interfaces,* vol. 28, pp. 670–694, 2006.

[33] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *Communications Surveys & Tutorials, IEEE,* vol. 12, pp. 343-356, 2010.

[34] V. Engen, "Machine learning for network based intrusion detection: an investigation into discrepancies in findings with the KDD cup'99 data set and multi-objective evolution of neural network classifier ensembles from imbalanced data," Bournemouth University, 2010.

[35] E. Mjolsness and D. DeCoste, "Machine learning for science: state of the art and future prospects," *Science,* vol. 293, pp. 2051-2055, 2001.

[36] J. Han and Micheline, *Data Mining: Concepts and Techniques*, Second Edition ed., 2006.

[37] Hand D., Mannila H., and Smyth P, *Principles of Data Mining*. Cambridge: Massachusetts Institute of Technology, 2001.

[38] Jiawei Han, Micheline Kamber, and Jian Pei, *Data mining : concepts and techniques, 3rd ed.* 225 Wyman Street, Waltham, MA 02451, USA: Morgan Kaufmann Publishers is an imprint of Elsevier, 2012.

[39] M. M. Suarez-Alvarez, D.-T. Pham, M. Y. Prostov, and Y. I. Prostov, "Statistical approach to normalization of feature vectors and clustering of mixed datasets," in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 2012, p. rspa20110704.

[40] J. A. Suykens, *Advances in learning theory: methods, models, and applications* vol. 190 P. 391: IOS Press, 2003.

[41] D. L. Olson and D. Delen, *Advanced data mining techniques*: Springer Science & Business Media, 2008.

[42] DeComite F., Denis F., Gillerson R., and Letouzey F., "Positive and unlabeled examples help learning," *10th International Conference on Algorithmic Learning Theory.,* vol. 1720, 1999.

[43] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Communications Surveys & Tutorials, IEEE,* vol. 10, pp. 56-76, 2008.

[44] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita, "An effective unsupervised network anomaly detection method," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, 2012, pp. 533-539.

[45] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning intrusion detection: supervised or unsupervised?," in *Image Analysis and Processing–ICIAP 2005*, ed: Springer, 2005, pp. 50-57.

[46] S. M. Hameed and S. S. Sulaiman, "Intrusion Detection Using a Mixed Features Fuzzy Clustering Algorithm," *Iraq Journal of Science (IJS),* vol. 53, 2012.

[47] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, 2005, pp. 333-342.

[48] P. V. Amoli and T. Hamalainen, "Real time multi stage unsupervised intelligent engine for NIDS to enhance detection rate of unknown attacks," in *Information Science and Technology (ICIST), 2013 International Conference on*, 2013, pp. 702-706.

[49] J. Li, W. Zhang, and K. Li, "A Novel Semi-supervised SVM based on Tri-training for Intrusition Detection," *Journal of computers,* vol. 5, pp. 638-645, 2010.

[50] S. Jiang, X. Song, H. Wang, J.-J. Han, and Q.-H. Li, "A clustering-based method for unsupervised intrusion detections," *Pattern Recognition Letters,* vol. 27, pp. 802-810, 2006.

[51] A. K. Jain and R. C. Dubes, *Algorithms for clustering data* vol. 6: Prentice hall Englewood Cliffs, 1988.

[52] M. A. Rassam, M. Maarof, and A. Zainal, "A survey of intrusion detection schemes in wireless sensor networks," *American Journal of Applied Sciences,* vol. 9, pp. 1636-1652, 2012.

[53] X. Zhu, "Semi-supervised learning literature survey," *Computer Sciences Technical Report 1530, University of Wisconsin–Madison,* 2005.

[54] J. Wang, K. Zhang, and D.-s. Ren, "An anomaly intrusion detection algorithm based on minimal diversity semi-supervised clustering," in *Computer Science and Computational Technology, 2008. ISCSCT'08. International Symposium on*, 2008, pp. 525-528.

[55] T. T. Lu, "Fundamental limitations of semi-supervised learning," *M.S. thesis, Dept. of Comput. Sci., Univ. of Waterloo, Waterloo, ON, Canada,* 2009.

[56] K.-L. Li, H.-K. Huang, S.-F. Tian, and W. Xu, "Improving one-class SVM for anomaly detection," in *Machine Learning and Cybernetics, 2003 International Conference on*, 2003, pp. 3077-3081.

[57] S. Araki, Y. Yamaguchi, H. Shimada, and H. Takakura, "Unknown Attack Detection by Multistage One-Class SVM Focusing on Communication Interval," in *Neural Information Processing*, 2014, pp. 325-332.

[58] P. Winter, E. Hermann, and M. Zeilinger, "Inductive intrusion detection in flow-based network data using one-class support vector machines," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, 2011, pp. 1-5.

[59] G. Giacinto, R. Perdisci, M. Del Rio, and F. Roli, "Intrusion detection in computer networks by a modular ensemble of one-class classifiers," *Information Fusion,* vol. 9, pp. 69-82, 2008.

[60] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol--HTTP/1.1," 2070-1721, 1999.

[61] D. Tamara, "Network+ Guide to Networks . Boston: Cengage Learning," ISBN 978-1-4239-0245-42010.

[62] Forouzan and Behrouz A., *Data Communications And Networking, Fourth ed. Boston: McGraw-Hill. pp. 621–630. ISBN 0-07-296775-7*, 2007.

[63] J. Postel, "Internet control message protocol.," 1981.

[64] MIT Lincoln Lab. (1999, 10/08/2015). *KDD Cup'99 Intrusion Attacks Database*. Available: http://www.ll.mit.edu/ideval/docs/attackDB.html

[65] N. Chandolikar and V. Nandavadekar, "Selection of Relevant Feature for Intrusion Attack Classification by Analyzing KDD Cup 99," *MIT International Journal of Computer Science & Information Technology,* vol. 2, pp. 85-90, 2012.

[66] A. Almutairi and D. Parish, "Using Classification Techniques for Creation of Predictive Intrusion Detection Model," *9th International Conference for Internet Technology and Secured Transactions (ICITST-2014),* 2014.

[67] L. Xie, D. Zhu, and H. Yang, "Research on SVM based network intrusion detection classification," in *Fuzzy Systems and Knowledge Discovery, 2009. FSKD'09. Sixth International Conference on*, 2009, pp. 362-366.

[68] M.-Y. Su, "Using clustering to improve the KNN-based classifiers for online anomaly network traffic identification," *Journal of Network and Computer Applications,* vol. 34, pp. 722-730, 2011.

[69] Christine Dartigue, Hyun Ik Jang, and Wenjun Zeng, "A New Data-Mining Based Approach for Network Intrusion Detection," *Seventh Annual Communications Networks and Services Research Conference,* 2009.

[70] Mohamed M. Abd-Eldayem, "A proposed HTTP service based IDS," *Egyptian Informatics Journal,* vol. Vol. 15 P.13–24, 2014.

[71] LI Han, "Research of K-MEANS Algorithm based on Information Entropy in Anomaly Detection," *Fourth International Conference on Multimedia Information Networking and Security,* 2012.

[72] Kyoto2006+, "Dataset, http://www.takakura.com/Kyoto_data/," 2009.

[73] J. Ma and G. Dai, "Anomaly detection in computer networks using dissimilarity-based one-class classifiers," in *Intelligent Systems Design and Applications, 2008. ISDA'08. Eighth International Conference on*, 2008, pp. 14-18.

[74] Anazida Zainal, Mohd Aizaini Maarof, and Siti Mariyam Shamsuddin, "Ensemble Classifiers for Network Intrusion Detection System," *Journal of Information Assurance and Security,* vol. Vol. 4 p. 217-225, 2009.

[75] C. Chen, Y. Gong, and Y. Tian, "Semi-supervised learning methods for network intrusion detection," in *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, 2008, pp. 2603-2608.

[76] S. K. Wagh and S. R. Kolhe, "Effective intrusion detection system using semi-supervised learning," in *Data Mining and Intelligent Computing (ICDMIC), 2014 International Conference on*, 2014, pp. 1-5.

[77] H. Yang, F. Xie, and Y. Lu, "Research on network anomaly detection based on clustering and classifier," in *Computational Intelligence and Security, 2006 International Conference on*, 2006, pp. 592-597.

[78] Y. Li, Z. Li, and R. Wang, "Intrusion detection algorithm based on semi-supervised learning," in *Information Technology, Computer Engineering and Management Sciences (ICM), 2011 International Conference on*, 2011, pp. 153-156.

[79] V. S. Mahajan and B. Verma, "Implementation of network traffic classifier using semi supervised machine learning approach," in *Engineering (NUiCONE), 2012 Nirma University International Conference on*, 2012, pp. 1-6.

[80] S. Sanfilippo. (2005, 02/06/2015). *Hping3 Security Tool*. Available: http://www.hping.org/download.html

[81] J. Hutchens. (2014). *SockStress Testing Tool : Proof of Concept*. Available: https://github.com/hack1thu7ch/Python-SockStress

[82] S. Shekyan. (2013, 23/07/2015). *Slowhttptest Testing Tool v1.6*. Available: https://code.google.com/p/slowhttptest/downloads/list

[83] Free Software Foundation. (2010, 03/05/2015). *jNetPcap Package Version 1.4.r1425*. Available: http://jnetpcap.com/download

[84] The Apache Software Foundation. (2015, 03/05/2015). *Apache HttpComponents Client version 4.4.1*. Available: https://www.apache.org/dist/httpcomponents/httpclient/binary/httpcomponents-client-4.4.1-bin.zip

[85] K. L. Ingham and H. Inoue, "Comparing anomaly detection techniques for http," in *Recent Advances in Intrusion Detection*, 2007, pp. 42-62.

[86] C. Torrano-Giménez, A. Perez-Villegas, and G. Álvarez Marañón, "An anomaly-based approach for intrusion detection in web traffic," 2010.

[87] H. S. V. Javitz, A., "The NIDES statistical component: Description and justication.," *Technical report, SRI International.,* 1993.

[88] D. Denning, "An intrusion detection model," *In IEEE Transactions on Software Engineering 13.,* 1987.

[89] H.-P. Kriegel, P. Kröger, and A. Zimek, "Outlier detection techniques," in *Tutorial at the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2009.

[90] A. A. Olusola, A. S. Oladele, and D. O. Abosede, "Analysis of KDD'99 Intrusion detection dataset for selection of relevance features," in *Proceedings of the World Congress on Engineering and Computer Science*, 2010, pp. 20-22.

[91] Z. Pawlak, *Rough sets: Theoretical aspects of reasoning about data* vol. 9: Springer Science & Business Media, 2012.

[92] S. V. Stehman, "Selecting and interpreting measures of thematic classification accuracy," *Remote sensing of Environment,* vol. 62, pp. 77-89, 1997.

[93] RapidMiner. (14/07/2015). *RapidMiner Studio 6.0*. Available: https://rapidminer.com/products/studio/

[94] Oracle. (30/04/2015). *Oracle Database 10g*. Available: http://docs.oracle.com/cd/B10500_01/index.htm

# Appendix A: Model Proof of concept's code

```
/**
 This function is used to retrieve the standard deviation of normal class
**/
CREATE OR REPLACE FUNCTION get_AUN_normal_std_dev RETURN NUMBER AS
-- Author: Ramzi A.M Mata
total_record  number;
std_dev number;
BEGIN
  select count(*) into total_record from AUN_FLOW_CLUSTER_NORMAL_TB;

  select sqrt(sum(power(sqr_distance,2))/(total_record-1)) into std_dev
from(
  SELECT
       sqrt(
          power(n.CLIENT_REPLY_ACK_TRUE - C.CLIENT_REPLY_ACK_TRUE,2)
        + power(n.CLIENT_REPLY_ACK_FALSE - C.CLIENT_REPLY_ACK_FALSE,2)
        + power(n.NUMBER_OF_SERVER_ACK - C.NUMBER_OF_SERVER_ACK ,2)
        + power(n.IS_HTTP_SESSION_TRUE - C.IS_HTTP_SESSION_TRUE,2)
        + power(n.IS_HTTP_SESSION_FALSE - C.IS_HTTP_SESSION_FALSE,2)
        + power(n.IS_HTTP_HEADER_END_TRUE - C.IS_HTTP_HEADER_END_TRUE,2)
        + power(n.IS_HTTP_HEADER_END_FALSE - C.IS_HTTP_HEADER_END_FALSE,2)
        + power(n.AVG_TIME_HTTP_HEADER_COMPLETE -
C.AVG_TIME_HTTP_HEADER_COMPLETE ,2)

      + power(n.NUMBER_OF_CLIENT_TCP_PSH - C.NUMBER_OF_CLIENT_TCP_PSH ,2)
      + power(n.AVG_TCP_PAYLOAD_LENGTH  - C.AVG_TCP_PAYLOAD_LENGTH ,2)
      + power(n.AVG_CLNT_TCP_WINDOW_SIZE - C.AVG_CLNT_TCP_WINDOW_SIZE ,2)
      + power(n.AVG_CURRENT_CONNECTIONS_4SEC -
C.AVG_CURRENT_CONNECTIONS_4SEC ,2)

      + power(n.AVG_USER_AGENTS_2SEC - C.AVG_USER_AGENTS_2SEC,2)
      + power(n.NUMBER_ZERO_WINDOW_PKTS - c.NUMBER_ZERO_WINDOW_PKTS,2)

  )  as sqr_distance
  from AUN_FLOW_CLUSTER_NORMAL_TB n, AUN_FLOW_CLUSTER_CENTROID_TB c
  );

  return std_dev;

END;
/

/**
 This function is used to test the model **/
CREATE OR REPLACE FUNCTION test_AUN_data(TUNE_V NUMBER) return number AS

std_dev number;
true_chk_v number;
false_chk_v number;
tmp_v number;
BEGIN

    std_dev := get_AUN_normal_std_dev;

FOR lbl in ( select distinct CLASS_TYPE from AUN_FLOW_CLUSTER_TEST_TB) loop
    true_chk_v:=0;
    false_chk_v:=0;

    for xx in (
              select count(std) as cnt,std from(
                    select distance, case
                          when distance> std_dev + tune_v then 0
                          else 1 end as std from(
```

```
                SELECT  CLASS_TYPE,
       sqrt(
          power(n.CLIENT_REPLY_ACK_TRUE - C.CLIENT_REPLY_ACK_TRUE,2)
        + power(n.CLIENT_REPLY_ACK_FALSE - C.CLIENT_REPLY_ACK_FALSE,2)
        + power(n.NUMBER_OF_SERVER_ACK - C.NUMBER_OF_SERVER_ACK ,2)
        + power(n.IS_HTTP_SESSION_TRUE - C.IS_HTTP_SESSION_TRUE,2)
        + power(n.IS_HTTP_SESSION_FALSE - C.IS_HTTP_SESSION_FALSE,2)
        + power(n.IS_HTTP_HEADER_END_TRUE - C.IS_HTTP_HEADER_END_TRUE,2)
        + power(n.IS_HTTP_HEADER_END_FALSE - C.IS_HTTP_HEADER_END_FALSE,2)

      + power(n.NUMBER_OF_CLIENT_TCP_PSH - C.NUMBER_OF_CLIENT_TCP_PSH ,2)
      + power(n.AVG_TCP_PAYLOAD_LENGTH  - C.AVG_TCP_PAYLOAD_LENGTH ,2)
      + power(n.AVG_CLNT_TCP_WINDOW_SIZE - C.AVG_CLNT_TCP_WINDOW_SIZE ,2)
      + power(n.AVG_CURRENT_CONNECTIONS_4SEC -
C.AVG_CURRENT_CONNECTIONS_4SEC ,2)

      + power(n.AVG_USER_AGENTS_2SEC - C.AVG_USER_AGENTS_2SEC,2)
      + power(n.NUMBER_ZERO_WINDOW_PKTS - c.NUMBER_ZERO_WINDOW_PKTS,2)

    )  as distance
  from AUN_FLOW_CLUSTER_TEST_TB n, AUN_FLOW_CLUSTER_CENTROID_TB c
                  where n.CLASS_TYPE=lbl.CLASS_TYPE
           )
      ) group by  std
 ) loop

       if xx.std = 1 then
            false_chk_v:=xx.cnt;
         else
          true_chk_v:=xx.cnt;
       end if;

 end loop;
     if lbl.CLASS_TYPE='Normal' then
         tmp_v:=false_chk_v;
         false_chk_v:=true_chk_v;
         true_chk_v:=tmp_v;
     end if;

     INSERT INTO AUN_TEST_RESULT_TB (LABEL, TRUE_CHECK, FALSE_CHECK,
TOTAL_RECORD, ACCURACY,tune)
     VALUES (lbl.CLASS_TYPE, true_chk_v, false_chk_v,
false_chk_v+true_chk_v, true_chk_v/(false_chk_v+true_chk_v),tune_v);

   end loop;

 commit;

 return 0;
END;
/
```